**Isolation Levels and Concurrency**

* [Isolation Levels and Concurrency](http://db.apache.org/derby/docs/10.0/manuals/develop/develop71.html#Isolation+Levels+and+Concurrency)

**Isolation Levels and Concurrency**

Derby provides four transaction isolation levels. Setting the transaction isolation level for a connection allows a user to specify how severely the user's transaction should be isolated from other transactions. For example, it allows you to specify whether transaction A is allowed to make changes to data that have been viewed by transaction B before transaction B has committed.

A connection determines its own isolation level, so JDBC provides an application with a way to specify a level of transaction isolation. It specifies four levels of transaction isolation. The higher the transaction isolation, the more care is taken to avoid conflicts; avoiding conflicts sometimes means locking out transactions. Lower isolation levels thus allow greater concurrency.

Inserts, updates, and deletes always behave the same no matter what the isolation level is. Only the behavior of select statements varies.

To set isolation levels you can use the JDBC *Connection.setIsolationLevel* method or the SQL SET ISOLATION statement. The names of the isolation levels are different, depending on whether you use a JDBC method or SQL statement. [Table 3](http://db.apache.org/derby/docs/10.0/manuals/develop/develop71.html#TBLSII-CONCEPTS-ISO_LEVEL) shows the equivalent names for isolation levels whether they are set through the JDBC method or an SQL statement.

**Table 3. Mapping of JDBC transaction isolation levels to Derby isolation levels**

|  |  |
| --- | --- |
| **Isolation levels for JDBC** | **Isolation levels for SQL** |
| Connection.TRANSACTION\_READ\_ UNCOMMITTED (ANSI level 0) | UR, DIRTY READ, READ UNCOMMITTED |
| Connection.TRANSACTION\_READ\_COMMITTED (ANSI level 1) | CS, CURSOR STABILITY, READ COMMITTED |
| Connection.TRANSACTION\_REPEATABLE\_READ (ANSI level 2) | RS |
| Connection.TRANSACTION\_SERIALIZABLE (ANSI level 3) | RR, REPEATABLE READ, SERIALIZABLE |

These levels allow you to avoid particular kinds of transaction anomalies, which are described in [Table 4](http://db.apache.org/derby/docs/10.0/manuals/develop/develop71.html#TBLSII-CONCEPTS-17741).

**Table 4. Transaction Anomalies**

|  |  |
| --- | --- |
| **Anomaly** | **Example** |
| Dirty Reads  A dirty read happens when a transaction reads data that is being modified by another transaction that has not yet committed. | Transaction A begins.  UPDATE employee SET salary = 31650  WHERE empno = '000090'  Transaction B begins.  SELECT \* FROM employee  (Transaction B sees data updated by transaction A. Those updates have not yet been committed.) |
| Non-Repeatable Reads  Non-repeatable reads happen when a query returns data that would be different if the query were repeated within the same transaction. Non-repeatable reads can occur when other transactions are modifying data that a transaction is reading. | Transaction A begins.  SELECT \* FROM employee  WHERE empno = '000090'  Transaction B begins.  UPDATE employee SET salary = 30100  WHERE empno = '000090'  (Transaction B updates rows viewed by transaction A before transaction A commits.) If Transaction A issues the same SELECT statement, the results will be different. |
| **Phantom Reads**  **Records that appear in a set being read by another transaction. Phantom reads can occur when other transactions insert rows that would satisfy the WHERE clause of another transaction's statement.** | **Transaction A begins.**  **SELECT \* FROM employee**  **WHERE salary > 30000**  **Transaction B begins.**  **INSERT INTO employee**  **(empno, firstnme, midinit,**  **lastname, job,**  **salary) VALUES ('000350', 'NICK',**  **'A','GREEN','LEGAL COUNSEL',35000)**  **Transaction B inserts a row that would satisfy the query in Transaction A if it were issued again.** |

The transaction isolation level is a way of specifying whether these transaction anomalies are allowed. The transaction isolation level thus affects the quantity of data locked by a particular transaction. In addition, a DBMS's locking schema might also affect whether these anomalies are allowed. A DBMS can lock either the entire table or only specific rows in order to prevent transaction anomalies.

[Table 5](http://db.apache.org/derby/docs/10.0/manuals/develop/develop71.html#TBLSII-CONCEPTS-28076) shows which anomalies are possible under the various locking schemas and isolation levels.

**Table 5. When Transaction Anomalies Are Possible**

|  |  |  |
| --- | --- | --- |
| **Isolation Level** | **Table-Level Locking** | **Row-Level Locking** |
| TRANSACTION\_READ\_UNCOMMITTED | Dirty reads, nonrepeatable reads, and phantom reads possible | Dirty reads, nonrepeatable reads, and phantom reads possible |
| TRANSACTION\_READ\_COMMITTED | Nonrepeatable reads and phantom reads possible | Nonrepeatable reads and phantom reads possible |
| TRANSACTION\_REPEATABLE\_READ | Phantom reads not possible because entire table is locked | Phantom reads possible |
| TRANSACTION\_SERIALIZABLE | None | None |

The following *java.sql.Connection* isolation levels are supported:

* TRANSACTION\_SERIALIZABLE

RR, SERIALIZABLE, or REPEATABLE READ from SQL.

*TRANSACTION\_SERIALIZABLE* means that Derby treats the transactions as if they occurred serially (one after the other) instead of concurrently. Derby issues locks to prevent all the transaction anomalies listed in [Table 4](http://db.apache.org/derby/docs/10.0/manuals/develop/develop71.html#TBLSII-CONCEPTS-17741) from occurring. The type of lock it issues is sometimes called a *range lock*. For more information about range locks, see [Scope of Locks](http://db.apache.org/derby/docs/10.0/manuals/develop/develop74.html#HDRCONCEPTS842524).

* TRANSACTION\_REPEATABLE\_READ

RS from SQL.

*TRANSACTION\_REPEATABLE\_READ* means that Derby issues locks to prevent only dirty reads and non-repeatable reads, but not phantoms. It does not issue range locks for selects.

* TRANSACTION\_READ\_COMMITTED

CS or CURSOR STABILITY from SQL.

*TRANSACTION\_READ\_COMMITTED* means that Derby issues locks to prevent only dirty reads, not all the transaction anomalies listed in [Table 4](http://db.apache.org/derby/docs/10.0/manuals/develop/develop71.html#TBLSII-CONCEPTS-17741).

*TRANSACTION\_READ\_COMMITTED* is the default isolation level for transactions.

* TRANSACTION\_READ\_UNCOMMITTED

UR, DIRTY READ, or READ UNCOMMITTED from SQL.

For a SELECT INTO, FETCH with a read-only cursor, full select used in an INSERT, full select/subquery in an UPDATE/DELETE, or scalar full select (wherever used), READ UNCOMMITTED allows:

* + Any row that is read during the unit of work to be changed by other application processes.
  + Any row that was changed by another application process to be read even if the change has not been committed by the application process.

For other operations, the rules that apply to READ COMMITTED also apply to READ UNCOMMITTED.