Verpflichtung der Implementierungen:

„The execution of concurrent SQL-transactions at isolation level SERIALIZABLE is guaranteed to be serializable. A serializable execution is defined to be an execution of the operations of concurrently executing SQL-transactions that produces the same effect as some serial execution of those same SQL-transactions. A serial execution is one in which each SQL-transaction executes to completion before the next SQL-transaction begins.“

Das Zitat stammt aus: Database Language SQL Document ISO/IEC 9075:1992 DIN 66315 Beuth Verlag Berlin Seite 54

Definition „serialisierbar“:

“Werden alle Transaktionen im Isolationsgrad SERIALIZABLE durchgeführt, dann wird die verzahnte Ausführung einer beliebigen Menge nebenläufiger Transaktion als serialisierbar in dem Sinne garantiert, dass derselbe Effekt wie bei einer (unbestimmten) seriellen Ausführung derselben Transaktionen nacheinander erzielt würde. Wird allerdings eine Transaktion in einem niedrigeren Isolationsgrad ausgeführt, kann die Serialisierbarkeit durch eine Reihe verschiedener Möglichkeiten verletzt werden. Der Standard definiert drei solche Möglichkeiten: Dirty Read, Nonrepeatable Read und Phantomzeilen...”

Das Zitat stammt aus: SQL – Der Standard SQL/92 mit den Erweiterungen CLI und PSM Deutsche Ausgabe des amerikanischen Klassikers Ausblick auf SQL3 Chris J. Date Hugh Darwen Addison Wesley Longman GmbH, 1998 Seite 85

5.5.2 ORACLE's ISOLATION LEVEL SERIALIZABLE „garantiert serialisierbar nicht“

Trans1	Zeitpunkt	Trans2
Oracle SET AUTOCOMMIT OFF; SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;		
Lese PK = 5 gefunden	t1	
	t2	Lese FK=5 nicht gefunden
INSERT FK=5 COMMIT	t3	
	t4	DELETE PK=5

Wir nehmen an, dass die Primärschlüssel/Fremdschlüsselbeziehung nicht implementiert ist!

Selbst wenn beide Transaktionen mit Oracle’s ISOLATION LEVEL SERIALIZABLE arbeiten entspricht das Ergebnis keinem serialisierten Ablauf!

Eingabe bei GOOGLE: ORACLE SERIALIZABLE 11

Zitat Anfang (*Hervorhebung nicht in der Originalquelle*)

Oracle9*i* Application Developer's Guide – Fundamentals
Release 2 (9.2)
Part Number A96590-01
Referential Integrity and Serializable Transactions
Because Oracle does not use read locks, even in SERIALIZABLE transactions, data read by one transaction can be overwritten by another. Transactions that perform database consistency checks at the application level should not assume that the data they read will not change during the execution of the transaction (even though such changes are not visible to the transaction). ***Database inconsistencies can result unless such application-level consistency checks are coded carefully, even when using SERIALIZABLE transactions.***
…The read issued by transaction A does not prevent transaction B from deleting the parent row, and transaction B's query for child rows does not prevent transaction A from inserting child rows. This scenario leaves a child row in the database with no corresponding parent row. This result occurs even if both A and B are SERIALIZABLE transactions, because neither transaction prevents the other from making changes in the data it reads to check consistency.
As this example shows, sometimes you must take steps to ensure that the data read by one transaction is not concurrently written by another. ***This requires a greater degree of transaction isolation than defined by SQL92 SERIALIZABLE mode.***

Zitat Ende

Die Implementierung von SERIALIZABLE bei ORACLE **erfüllt nicht** die Forderung des SQL Standards („The execution of concurrent SQL-transactions at isolation level SERIALIZABLE is guaranteed to be serializable”).

Vergleichen Sie dazu bitte auch die **irreführende Behauptung** (“This requires a greater degree of transaction isolation than defined by SQL92 *SERIALIZABLE* mode”) aus dem Developer’s Guide.

ORACLE Philosophie

- der Leser wartet nicht auf den Schreiber
- der Schreiber wartet nicht auf den Leser

Realisierung der Philosophie mit Multi Version Consistency Modell

statement-level read consistency als Default (file init.ora mit default parametern), d.h. es wird ein Snapshot zum Zeitpunkt des Query-Beginns gebildet, also nur committete Daten zu Query-Beginn sind sichtbar! Zur Gewährleistung der Datenkonsistenz auf Anweisungsebene benutzt ORACLE Ringpuffer in Rollback-Segmenten, die entsprechend konfiguriert sein müssen.

Außerdem wird ein ORACLE spezifischer Read-Only-Modus angeboten, dieser ermöglicht **transaction-level read consistency**.

ORACLE kennt

- SET TRANSACTION READ ONLY
- SET TRANSACTION ISOLATION LEVEL READ COMMITTED
- SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

Read-only Read-only transactions see only those changes that were committed at the time the transaction began and do not allow INSERT, UPDATE, and DELETE statements.

Read committed This is the default transaction isolation level. Each query executed by a transaction sees only data that was committed before the query (not the transaction) began. An Oracle Database query never reads dirty (uncommitted) data.

Because Oracle Database does not prevent other transactions from modifying the data read by a query, that data can be changed by other transactions between two executions of the query. Thus, a transaction that runs a given query twice can experience both nonrepeatable read and phantoms. **UPDATE und DELETE sind möglich gegen alle Zeilen die „committed“ sind, also gegen alle Zeilen die nicht von einer anderen Transaktion gesperrt sind.**

Serializable Serializable transactions see only those changes that were committed at the time the transaction began, plus those changes made by the transaction itself through INSERT, UPDATE, and DELETE statements. Serializable transactions do not experience nonrepeatable reads or phantoms. **UPDATE und DELETE sind nur möglich gegen Zeilen die zu Beginn der Transaction „committed“ waren.**

5.5.3 SQL Server's ISOLATION LEVEL SERIALIZABLE „garantiert serialisierbar“

Trans1
SQL Server
`SET IMPLICIT_TRANSACTIONS ON;`
`SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;`
	Zeitpunkt	Trans2
		SQL Server
		`SET IMPLICIT_TRANSACTIONS ON;`
		`SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;`

Lese PK = 5 gefunden S-Lock	t1	
	t2	Lese FK=5 nicht gefunden
INSERT FK=5 Wait	t3	
Wait Wait Wait	t4	 DELETE PK=5 Wait

Wir nehmen an, dass die Primärschlüssel/Fremdschlüsselbeziehung nicht implementiert ist!

Wenn beide Transaktionen mit SQL Server ISOLATION LEVEL SERIALIZABLE arbeiten entspricht das Ergebnis einem serialisierten Ablauf!

Es kommt zum Deadlock, eine der beiden Anweisungen INSERT bzw. DELETE wird nicht erfolgreich ausgeführt.

SQL Server ISOLATION LEVEL SNAPSHOT

Mit Hilfe von ALTER DATABASE kann die Snapshotisolation aktiviert werden.

```
ALTER DATABASE dbbuecher
    SET ALLOW_SNAPSHOT_ISOLATION ON;
```

Beachten Sie bitte, dass in der Sekundärliteratur Mythen verbreitet werden der Art: „SQL Server adds SNAPSHOT isolation that, in effect, provides alternate implementations of SERIALIZABLE ...“.

Wenn beide Transaktionen mit SQL Server ISOLATION LEVEL SNAPSHOT arbeiten entspricht das Ergebnis **keinem serialisierten Ablauf**!

Vergleichen Sie dazu bitte auch die Diskussion zu den Besonderheiten von ORACLE’s Implementierung.

5.5.4 DB2's Isolation Level RR „garantiert serialisierbar“

Trans1	Zeitpunkt	Trans2

DB2 for Windows, UNIX, Linux
`UPDATE COMMAND OPTIONS USING C OFF;`
`CHANGE ISOLATION TO RR;`
		DB2 for Windows, UNIX, Linux
		`UPDATE COMMAND OPTIONS USING C OFF;`
		`CHANGE ISOLATION TO RR;`

Lese PK = 5 gefunden S-Lock	t1	
	t2	Lese FK=5 nicht gefunden
INSERT FK=5 Wait	t3	
Wait Wait Wait	t4	DELETE PK=5 Wait

Wir nehmen an, dass die Primärschlüssel/Fremdschlüsselbeziehung nicht implementiert ist!

Wenn beide Transaktionen mit DB2's Isolation Level RR arbeiten entspricht das Ergebnis einem serialisierten Ablauf!

Es kommt zum Deadlock, eine der beiden Anweisungen INSERT bzw. DELETE wird nicht erfolgreich ausgeführt.

5.5.5 Dirty Read – der schmutzige Read

Trans1	Zeitpunkt	Trans2
	t1	**UPDATE** **X-Lock** **exklusive Sperre** **X-Lock**
Oracle Ein schmutziges Lesen ist nicht möglich!		**X-Lock** **X-Lock**

SQL Server
`SET IMPLICIT_TRANSACTIONS ON;`
`SET TRANSACTION ISOLATION LEVEL READ UNCOMMITTED;`
 	
`SELECT ...`
 	
DB2 for Windows, UNIX, Linux
`UPDATE COMMAND OPTIONS USING C OFF;`
`CHANGE ISOLATION TO UR;`
 	
`SELECT ...`
`SELECT ... WITH UR;`
	
	t2
	
	t2
	t2
		X-Lock
		
		X-Lock

DB2 for z/OS SPUFI **„autocommit off“** `SELECT ... WITH UR;`	 t2	**X-Lock** **X-Lock**
	t3	**ROLLBACK**

"Transaktion Trans1 hat zum Zeitpunkt t2 Zugang zu nicht festgeschriebenen Daten"

Parameter für Trans1 mit dem Ziel den Dirty Read zu vermeiden:

Oracle:

Bei Oracle ist auf Grund der “statement-level read consistency“ ein schmutziges Lesen nicht möglich!

SQL Server:

```
SET TRANSACTION ISOLATION LEVEL READ COMMITTED;
SET TRANSACTION ISOLATION LEVEL SNAPSHOT;
```

DB2 for Windows, UNIX, Linux:

```
CHANGE ISOLATION TO CS;
SELECT ... WITH CS;
```

DB2 for z/OS SPUFI:

“ISOLATION CS im SPUFI Default Panel”

```
SELECT ... WITH CS;
```

5.5.6 Lost Update – der verlorene Update

Das folgende Coding SELECT/UPDATE ist abhängig vom Isolation Level.

Trans1	Zeitpunkt	Trans2

Oracle
`SET AUTOCOMMIT OFF;`
`SET TRANSACTION ISOLATION LEVEL READ COMMITTED;`
SQL Server
`SET IMPLICIT_TRANSACTIONS ON;`
`SET TRANSACTION ISOLATION LEVEL READ COMMITTED;`
DB2 for Windows, UNIX, Linux
`UPDATE COMMAND OPTIONS USING C OFF;`
`CHANGE ISOLATION TO CS;`
DB2 for z/OS SPUFI
„autocommit off“
“ISOLATION CS im SPUFI Default Panel”		

`SELECT konto`
`FROM tkonten`
`WHERE kontonr = 5`
es sind 3.50
Lesen ohne Sperren
	t1	

erhöhe um 200.00	t2	`SELECT konto`
		`FROM tkonten`
		`WHERE kontonr = 5`
		es sind 3.50
		Lesen ohne Sperren

`UPDATE tkonten SET`
`konto = 203.50`
`WHERE kontonr = 5`
Schreiben ohne Prüfen erfolgreich!
	t3	ziehe 50.00 ab

COMMIT	t4	

	t5	`UPDATE tkonten SET`
		`konto = - 46.50`
		`WHERE kontonr = 5`
		Schreiben ohne Prüfen erfolgreich!
		COMMIT

"Transaktion Trans1 verliert zum Zeitpunkt t5 einen Update"

Was ist das Problem? Lesen ohne Sperren und Schreiben ohne Prüfen!

Von allen Beteiligten muss entweder beim Lesen gesperrt werden oder beim Schreiben geprüft werden!

Parameter für alle Transaktionen mit dem Ziel, den verlorenen Update zu vermeiden:

Oracle (Schreiben mit Prüfen):

Bei SERIALIZABLE wird vom System beim Schreiben geprüft.

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
```

Die Klausel **SELECT…FOR UPDATE OF…** bietet die Möglichkeit, eine gelesene Zeile zu sperren, auch unter der Voraussetzung **SET AUTOCOMMIT ON**.

SQL Server (Schreiben mit Prüfen oder Lesen mit Sperren):

Bei SERIALIZABLE und REPEATABLE READ wird vom System gesperrt.

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;
SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;
```

Bei SNAPSHOT wird vom System beim Schreiben geprüft.

```
SET TRANSACTION ISOLATION LEVEL SNAPSHOT;
```

Die Klausel SELECT…WITH(UPDLOCK) bietet die Möglichkeit, eine gelesene Zeile zu sperren.

DB2 (Lesen mit Sperren):

Bei RR und RS wird vom System gesperrt.

DB2 for Windows, UNIX, Linux:

```
CHANGE ISOLATION TO RR;
SELECT ... WITH RR;
CHANGE ISOLATION TO RS;
SELECT ... WITH RS;
```

DB2 for z/OS SPUFI:

"ISOLATION RR im SPUFI Default Panel"
"ISOLATION RS im SPUFI Default Panel"

```
SELECT ... WITH RR;
SELECT ... WITH RS;
```

Die Klausel SELECT…**FOR UPDATE OF…WITH RS** bietet die Möglichkeit, eine gelesene Zeile zu sperren.
ACHTUNG: *BEI* ***SELECT…FOR UPDATE…OF WITH CS*** *WIRD DIE ZEILE* ***NICHT GESPERRT****!*

Schreiben mit Prüfen im Coding, unabhängig vom Isolation Level

```
UPDATE  tkonten SET konto = 203.50
WHERE kontonr = 5 AND konto = 3.50;
```

Der verlorene Update ist nicht möglich, wenn eine andere, logisch nicht äquivalente Update-Anweisung benutzt wird.

```
UPDATE  tkonten SET konto = konto + 200.00
WHERE kontonr = 5;
```

Weitere Details (CURSOR FOR UPDATE, OPEN, FETCH, UPDATE WHERE CURRENT) sind für die Anwendungsentwicklung wichtig: wie müssen Update-Programme geschrieben werden, damit der LOST UPDATE nicht möglich ist!

5.5.7 Lesen mit Sperren, S-Lock, Deadlock

Das folgende Coding SELECT/UPDATE ist abhängig vom Isolation Level.

Trans1	Zeitpunkt	Trans2
SQL Server **SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;** SELECT konto FROM tkonten WHERE kontonr = 5; **es sind 3.50** **Lesen mit Sperren** **S-Lock**	t1	**SQL Server** **SET TRANSACTION ISOLATION LEVEL REPEATABLE READ;**
DB2 for Windows **CHANGE ISOLATION TO RS;** SELECT konto FROM tkonten WHERE kontonr = 5 **WITH RS;** **es sind 3.50** **Lesen mit Sperren** **S-Lock**	t1	**DB2 for Windows** **CHANGE ISOLATION TO RS;**
Erhöhe um 200.00	t2	SELECT konto FROM tkonten WHERE kontonr = 5; **es sind 3.50** **Lesen mit Sperren** **S-Lock**
UPDATE tkonten SET konto = 203.50 WHERE kontonr = 5; **X-Lock ?** **Wait**	t3	ziehe 50.00 ab
Wait **Wait**	t4	
Wait **Wait** **Wait** **Wait** **Wait** **Wait**	t5	UPDATE tkonten SET konto = – 46.50 WHERE kontonr = 5; **X-Lock ?** **Wait** **Wait**

Sofern das System bei allen Lesevorgängen mit einem S-Lock sperrt kommt es bei konkurrierenden Transaktionen zum **Deadlock.** Fachsprache Informatik: „bei nebenläufigen Transaktionen kommt es zur Verklemmung“.

5.5.8 Schreiben mit Prüfen, kann nicht serialisieren

Das folgende Coding SELECT/UPDATE ist abhängig vom Isolation Level.

Trans1	Zeitpunkt	Trans2
Oracle **SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;**		**Oracle** **SET TRANSACTION ISOLATION LEVEL SERIALIZABLE;**
Oracle SELECT konto FROM tkonten WHERE kontonr = 5 **es sind 3.50** **Lesen ohne Sperren**	t1	
Erhöhe um 200.00	t2	**Oracle** SELECT konto FROM tkonten WHERE kontonr = 5 **es sind 3.50** **Lesen ohne Sperren**
abhängig vom Isolation Level UPDATE tkonten SET konto = 203.50 WHERE kontonr = 5 **Oracle Schreiben mit Prüfen erfolgreich!**	t3	ziehe 50.00 ab
COMMIT		
		abhängig vom Isolation Level UPDATE tkonten SET konto = 203.50 WHERE kontonr = 5 **Oracle Schreiben mit Prüfen can't serialize!**

Bei „nicht sperrenden Systemen" wie z.B. Oracle muss bei diesem Coding zwingend der ISOLATION LEVEL SERIALIZABLE benutzt werden!