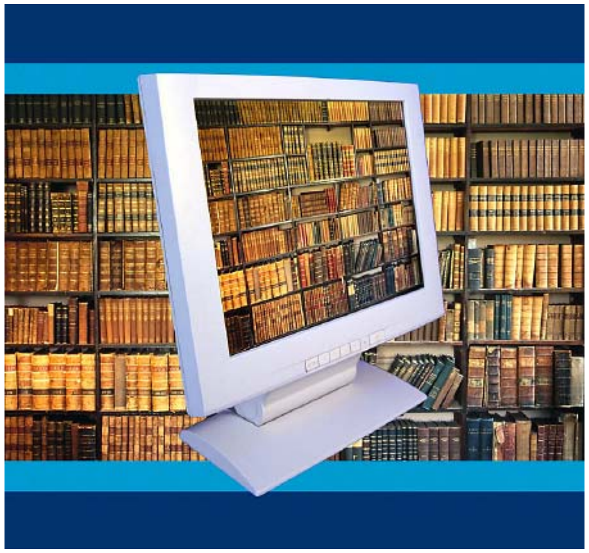
Database Management Today

Data Integrity in Operational Database Management Systems

DATA SUMMIT
UNLEASH THE POWER OF YOUR DATA

Data Summit 2017



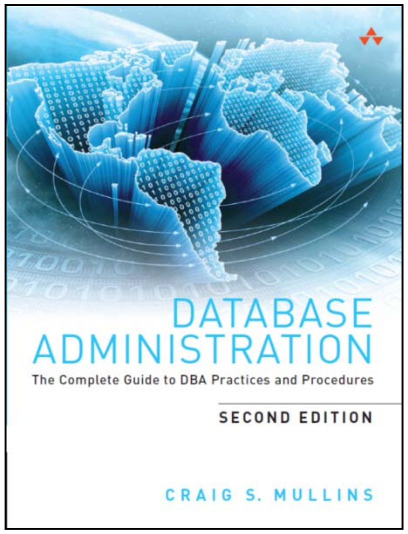
Craig S. Mullins
Mullins Consulting, Inc.
http://www.MullinsConsulting.com

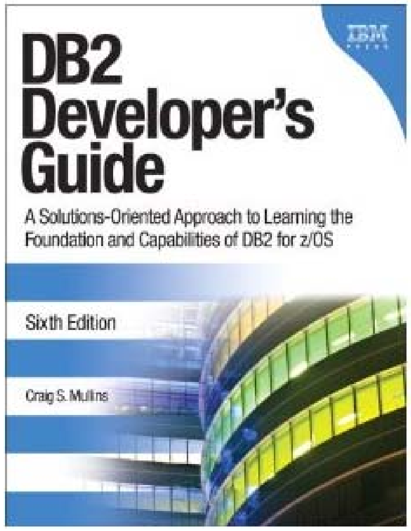
Author

This presentation was prepared by:

Craig S. Mullins
President & Principal Consultant

Mullins Consulting, Inc.
15 Coventry Ct
Sugar Land, TX 77479
Tel: 281-494-6153
Fax: 281.491.0637
Skype: cs.mullins
E-mail: craig@craigsmullins.com
http://www.mullinsconsultinginc.com




IBM
DB2 Developer's Guide
A Solutions-Oriented Approach to Learning the Foundation and Capabilities of DB2 for z/OS
Sixth Edition
Craig S. Mullins

Craig was named one of the Top 200 Thought Leaders in BigData & Analytics by AnalyticsWeek.

Agenda

Data Integrity in Operational Database Systems

- **What is meant by Data Integrity**
- **RDBMS vs. NoSQL**
- **ACID vs. BASE**
- **Locking**
- **Isolation Levels**
- **Types of Eventual Consistency**
- **Other Data Integrity Issues**
    - **Referential Integrity**

What is Data Integrity?

Data integrity in an operational DBMS is making sure that the data is always accurate and correct

- There are many aspects to data integrity
  - Correct in terms of internal DBMS functionality
    - Structure, pointers, page maps, etc.
  - Correct in terms of its business meaning
    - IQ vs. Shoe size
  - Correct in terms of the actual values of data elements
    - Domain accuracy, data type, length, uniqueness, etc.
  - Correct in terms of the relationship between the data elements
    - Are all references accurate?
  - Changes are correctly applied when requested
    - Transactions processed correctly and appropriately

Data Integrity: *an example*

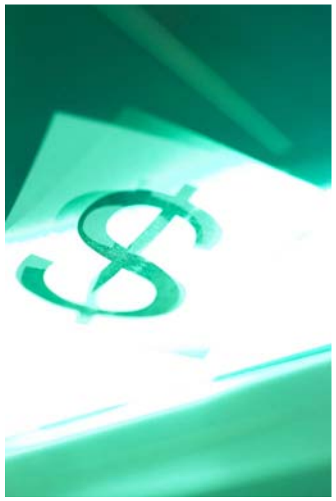
Joe goes to the bank to transfer $50 from his savings account to his checking account

- In Joe's mind, these things happen concurrently and immediately
- In practice, a program/transaction will perform these things in steps
  - Withdraw $50 from savings
  - Deposit $50 to checking
- But at no time is it correct to say that Joe has $50 less in his savings account if he does not also, at the same time, have $50 more in his checking account
  - The unit of work has to encompass both actions
  - If one fails, both must fail; if one succeeds both must succeed

The Banking Example *(continued)*

What happens if Joe's wife attempts to withdraw $20 from the same checking account at the same time that Joe is transferring funds?

- Let's say the checking account was down to $1 before Joe's transfer transaction began
- A database locking mechanism can prevent concurrent modifications to the same data
- This is important because:
  - What if Joe's transaction does not complete successfully?
  - If Joe's wife can withdraw $20 before Joe's transaction is committed, then the checking account will be overdrawn



Questions to be Answered

How do relational and NoSQL differ in terms of consistency?

Is ACID compliance as big of a deal as it is said to be?

How can data get compromised even with ACID compliance?

What impact can eventual consistency have on your data?

What about Referential Integrity?

Relational versus NoSQL – An Overview

Relational (e.g. DB2)	NoSQL
• Rigid, predefined schema	• Flexible schema
• Sound theoretical foundation (set theory)	• No theoretical foundation – many iterations & types
• Industry standard query language – SQL	• Query requires programming or add-on
• Two-dimensional row and column design	• Can accommodate complex data
• More difficult to scale (usually scale up)	• Elastic scaling (scale out)
• Usually commercial	• Usually open source
• ACID	• BASE (some provide ACID)

ACID versus BASE

ACID

- Atomicity (all or nothing)
- Consistency
  (data is always correct)
- Isolation (concurrent txns)
- Durability
  (data survives failures)

BASE

- Basically Available
  (no guarantee of data avialability; but the system will respond to any request)
- Soft State
  (changes constantly happening)
- Eventual Consistency

ACID: Atomicity

- ***Atomicity*** means that a transaction must exhibit an “all or nothing” behavior.
  - Either all of the instructions within the transaction happen, or none of them happen.
  - Atomicity preserves the “completeness” of the business process.

ACID: Consistency

- ***Consistency*** refers to the state of the data both before, and after, the transaction is executed.
  - A transaction maintains the consistency of the state of the data.
  - In other words, after running a transaction all data in the database is "correct."

ACID: Isolation

- ***Isolation*** means that transactions can run at the same time. Any transactions running in parallel have the illusion that there is no concurrency.
  - In other words, it appears that the system is running only a single transaction at a time.
  - No other concurrent transaction has visibility to the uncommitted database modifications made by any other transactions.
  - To achieve isolation a locking mechanism is required.
  - And there are various levels of isolation that can impact data integrity if you do not understand how they work and what they mean.

    Isolation level discussion forthcoming later in presentation…

ACID: Durability

- ***Durability*** refers to the impact of an outage or failure on a running transaction.
  - A durable transaction will not impact the state of data if the transaction ends abnormally.
  - The data will survive any failures.

BASE: Basically Available

- ***Basically Available*** means that there is no guarantee of any specific piece of data being available.
  - But the system will respond to any request
  - In other words, there can be a partial failure of a distributed system but the rest of the system will continue to operate
  - Consider a system with 10 servers. If one fails, the other nine continue to operate.
  - This means that data on the failing server will not be available…

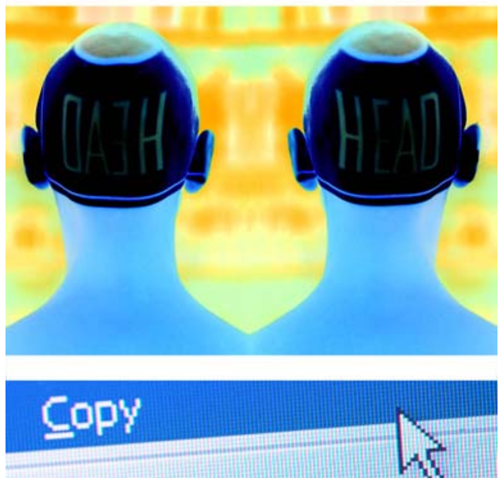
    Unless the data is replicated redundantly across multiple servers, which can provide failover relief

BASE: Soft State

- In a ***Soft State*** system changes are constantly happening. Of course, this is really no different than any active system. But
  - It means that the data you retrieve at a given point in time may eventually get over-written more recent data
  - Amazon seller example

BASE: Eventual Consistency

- ***Eventual Consistency*** means that there will be times when the database is in an inconsistent state.
  - When multiple copies of the data reside on separate servers (which is common in NoSQL databases), an update may not be immediately made to all copies simultaneously
  - So the data is inconsistent for a time,[1] but the database replication mechanism will eventually update all of the copies of the data to be consistent



[1] This is sometimes referred to as being "wrong" by relational DBAs.

ACID versus BASE

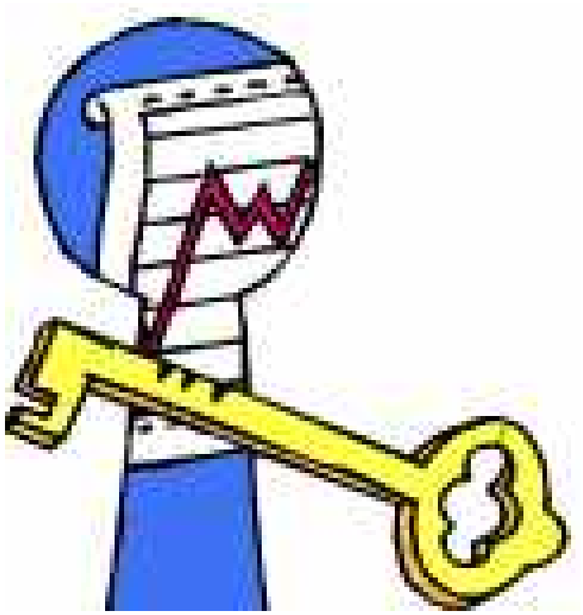
The bottom line is that both consistency methods can work depending upon the type of applications that you are writing.

- ACID lends itself to traditional transaction processing where the data must ***always*** be accurate ***at all times***
  - For example, banking
- BASE lends itself to situations where brief periods of inconsistency are tolerable
  - For example, web-based applications
- But you need to know what you are working with!
  - ACID does not "guarantee" data integrity!

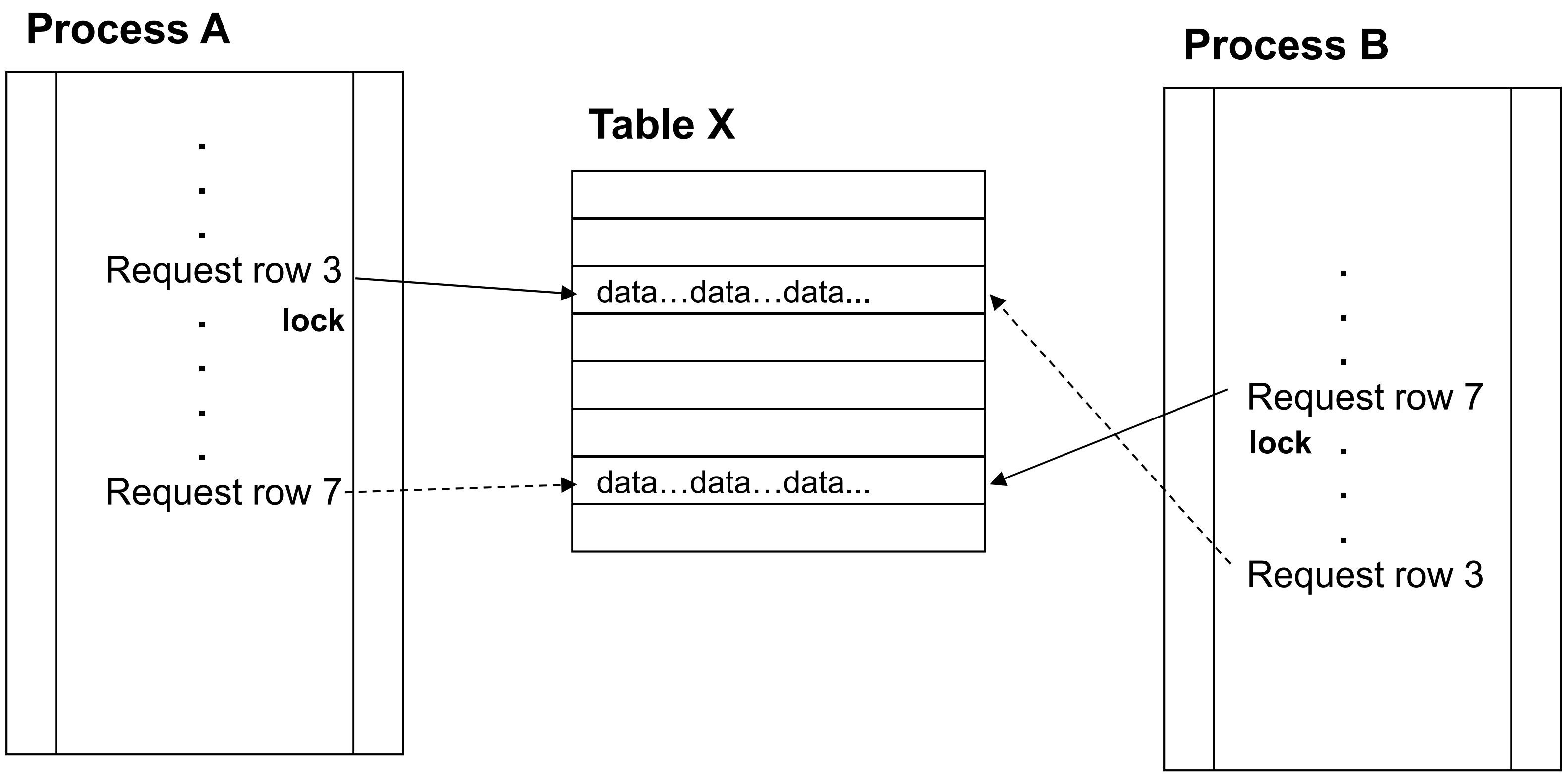
Locking is used with ACID to Implement Isolation

Locking

- Types of Locks
  - Share, eXclusive
  - Intent Locks
  - Row, Page, Table, Table Space
- Timeouts
- Deadlocks
  - Example on next page



Deadlocks



Process A is waiting on Process B

Process B is waiting on Process A

Lock Duration

Isolation Level

- This is the “I” in ACID
  - Read uncommitted
  - Read committed
  - Repeatable read
  - Serializable
- Specified for programs to dictate how the DBMS should isolate changes
  - The isolation or serialization parameter will have a big impact on the behavior of the program
  - Unfortunately, many folks are unaware of the impact…

Isolation: Read Uncommitted

Read Uncommitted isolation implements read-through locks and is sometimes referred to as a dirty read.

- It applies to read operations only. Eliminates read locking.
- Data may be read that never actually exists in the database, because the transaction can read data that has been changed by another process but is not yet committed.
  - A dirty read can cause duplicate rows to be returned where none exist or no rows may be returned when one (or more) actually exists.
- Read uncommitted isolation provides the highest level of availability and concurrency, but the worst data integrity.
- It should be used only when data integrity problems can be tolerated.
  - For example, analytical queries, estimates, and averages are likely candidates for read uncommitted locking.

Isolation: Read Committed

The Read Committed isolation level provides increased integrity control than read uncommitted.

- With read committed isolation, a transaction will never read data that is not yet committed
  - That is, **only committed data can be read**.
  - This lowers availability as reads on data that is being updated will be delayed (lock waits) until the modification is committed
- Sometimes referred to as Cursor Stability

Isolation: Repeatable Read

Repeatable Read place a further restriction on reads; the assurance that *the same data can be accessed multiple times during the transaction without its value changing*.

- The lower isolation levels permit the underlying data to change if it is accessed more than once.
- Write skew is still possible;
  - Two writes are allowed to the same column(s) in a row by two different writers (who have previously read the columns they are updating), resulting in the row having data that is a mix of the two transactions
- Use repeatable read when:
  - Your program reads the same data multiple different times during the course of the transaction and the data values must be consistent.

    Otherwise, processes can change data after it is read by your program; subsequent reads of "the same" data actually will **not** be the same.

    Phantoms can still exist, though… see Serializable (next slide)

Isolation: Serializable

The highest level of integrity is provided by the Serializable isolation level.

- Serializable isolation removes write skew and the possibility of phantoms.
- A ***phantom*** occurs when the transaction opens a cursor that retrieves data and subsequently another process inserts a value that would satisfy the request and should be in the result set.
  - Serializable isolation will ensure that reads issued more than once in a program will return the exact same data
    - No data inserted, updated or deleted by other programs allowed to data accessed by the program, while the program runs

Isolation Levels and Integrity

Isolation level	Dirty reads	Non-repeatable reads	Phantoms
Read Uncommitted	may occur	may occur	may occur
Read Committed	-	may occur	may occur
Repeatable Read	-	-	may occur
Serializable	-	-	-

Isolation Levels and Locking

Isolation level	Write	Read	Range
Read Uncommitted	S	S	S
Read Committed	C	S	S
Repeatable Read	C	C	S
Serializable	C	C	C

"C" - Locks are held until the transaction commits.
"S" - Locks are held only during the currently executing statement.

Questions?

Unless all of your programs that modify data are using the serializable isolation level data integrity issues can arise…

- Do you know what isolation level is used for all of your mission critical applications?

Isolation level can be controlled at the program or statement level…

- Do you know what actions your programmers are taking with regard to isolation?

Eventual Consistency

So we see that ACID is not just a "we support it" issue and all data is magically correct!

There are also different types of Eventual Consistency that can be supported

- Casual consistency
- Read-your-writes consistency
- Session consistency
- Monotonic read consistency
- Monotonic write consistency

Casual Consistency

Casual consistency maintains the order of updates.

- The database will reflect the order of modification operations performed on the data.
- Consider our banking example
  - Joe withdraws $50 from savings
  - Joe deposits $50 to checking
  - Joe's wife withdraws $20 from checking

Read-Your-Writes Consistency

Read-Your-Writes consistency means that when you modify data, all of your reads will return the modified value.

- Consider our banking example
  - Joe withdraws $50 from savings
  - Joe's wife queries the balance of checking

    The balance will not reflect the $50 as it has not been deposited yet
  - Joe's wife queries the balance of savings

    The balance will reflect the $50 that has been withdrawn

Session Consistency

Session consistency ensures Read-Your-Writes consistency at the session level.

- Consider our banking example
  - Joe withdraws $50 from savings
  - Joe deposits $50 to checking
  - Joe queries the balance of checking
    - The balance will reflect the $50 as it has been deposited
  - Joe queries the balance of savings
    - The balance will reflect the $50 that has been withdrawn
  - Joe ends his session
  - Joe queries his balance (of either account)
    - The balance may or may not reflect the deposit/withdrawal as another session has been initiated

Monotonic Read Consistency

Monotonic Read consistency ensures that if you issue a query and receive a result, you will never receive earlier values in subsequent queries.

- Consider our banking example
  - Joe withdraws $50 from savings
  - Joe deposits $50 to checking
  - Joe queries the balance of checking
    - The balance will reflect the $50 as it has been deposited
  - Joe queries the balance of checking again
    - The balance will reflect the $50 that has been deposited even if all servers with a copy of that data have not yet been updated

Monotonic Write Consistency

Monotonic Write consistency ensures the order of database modifications is maintained.

- Consider our banking example
  - Joe withdraws $50 from savings
  - Joe deposits $50 to checking
  - Joe's wife withdraws $20 from checking
  - Joe queries the balance of checking
    - The balance will reflect the $50 as it has been deposited
    - And it will also reflect the $20 that has been withdrawn by his wife
  - If the order of operations were not guaranteed then:
    - Joe's wife could withdraw the $20 before Joe's $50 deposit is recorded in the database
    - That means overdraft fees could be applied and when Joe queries the balance it will be lower
  - Monotonic Write consistency is important for consistent data!

Other Types of Data Integrity

Structural integrity

- Consistency Options
- Database Checking
- Memory Usage
- Additional Options

Semantic data integrity

- Entity Integrity
- Referential Integrity
  - User- vs. System-Managed RI
- Unique Constraints
- Data Types
- Default Values
- Check Constraints
- Triggers

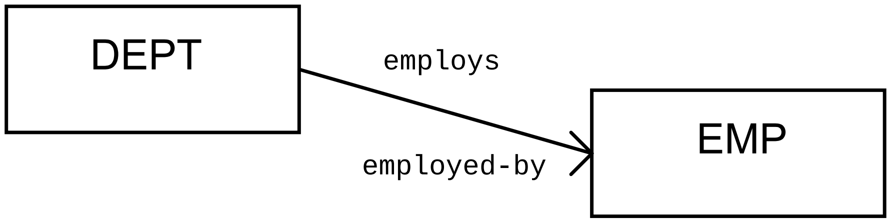
Referential Integrity

Primary Key

Foreign Key

DELETE RULE

- Restrict
- Cascade
- Set NULL



Managing Referential Sets

- Backup & Recovery
- LOAD
- CHECK

Summary

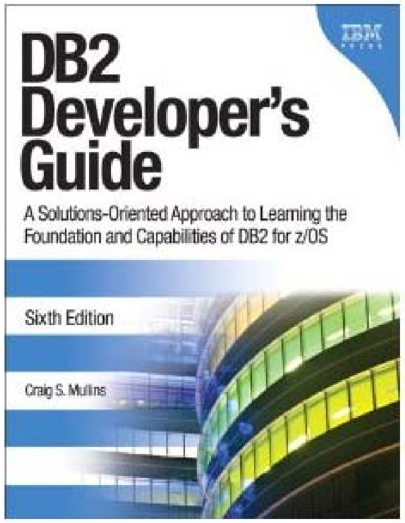
Data integrity and consistency is not a simple all or nothing proposition

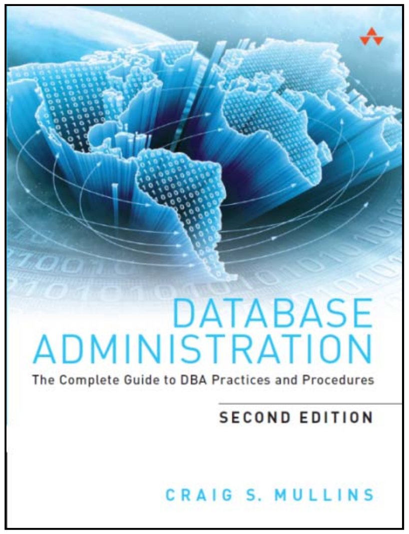
You cannot simply say “My DBMS supports ACID so my data is correct”

- You must understand what is supported and how
- You must educate developers to use the proper parameters
  - A DBMS with ACID where all programs use Uncommitted Read is not likely to result in data integrity
  - A DBMS with eventual consistency programmed appropriately can result in having reasonable data integrity
  - Know your application/business needs and proceed accordingly

We have looked at the basic issues but there are many additional nuances to data integrity that you will encounter

Contact Information


IBM Press
DB2 Developer's Guide
A Solutions-Oriented Approach to Learning the Foundation and Capabilities of DB2 for z/OS
Sixth Edition
Craig S. Mullins



Craig S. Mullins

Mullins Consulting, Inc.
15 Coventry Court
Sugar Land, TX 77479
Phone: (281) 494-6153

craig@craigsmullins.com

http://www.mullinsconsulting.com