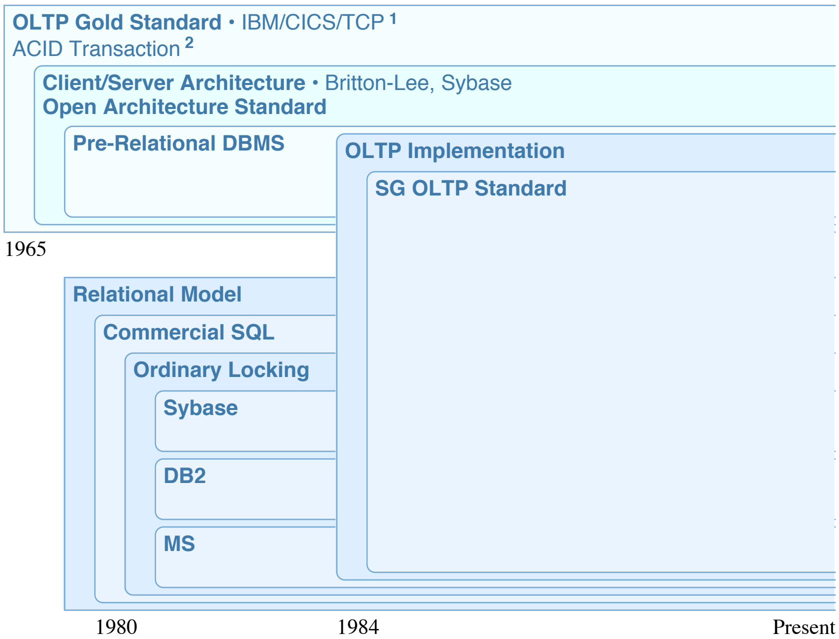
Reality vs Academic Isolation

Reality: Industry; Science; Actual Progress

This illustrates the appearance of specific technology under discussion, the progress of Transaction Processing in the real world, from its initial implementation to the present day. It is in chronological order (left to right, and an occasional scale with dates).



The entire progress in this science (not only Transaction Processing, but the entire database science) has been through platform vendors and their engineers, the genuine *cutting edge*. For decades. With the single exception of Dr E F Codd, who was and is, unsurprisingly, loved by the vendors who implemented his *Relational Model*, and by capable practitioners who did the same, but hated by academia.

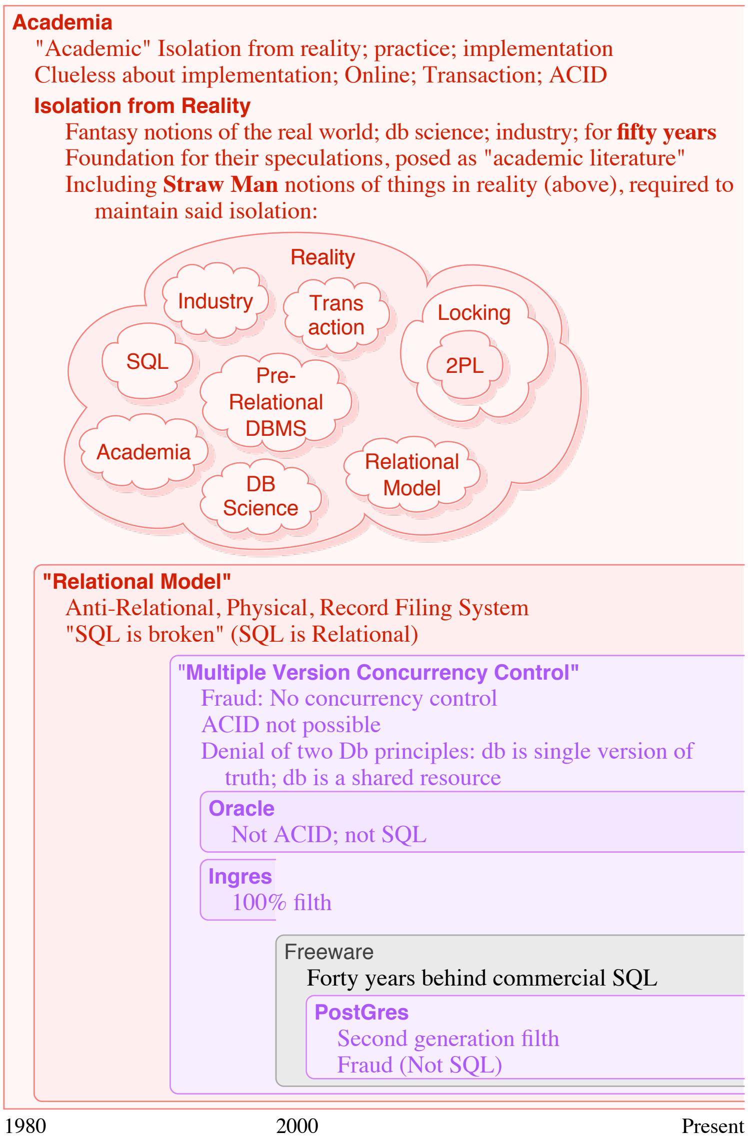
Academia fails totally, by divorcing themselves from Logic; science; implementation concerns, ensuring that they can never serve the industry that they claim to serve.

The most relevant point here is, they have no idea that Transaction Processing is a science, with implementation elements in both the database and the application. The Form has not changed since 1965 (truth does not change), the syntax has progressed merely to accommodate the progression of platforms.

The *Software Gems OLTP Standard* implements a high-performance OLTP context: lowest contention; highest concurrency, plus security layers that are integrated into the platform. It is not the subject of this paper, it is not expanded here.

Academia: Isolation; Ignorance; Pseudo-science; Fantasy

This illustrates the appearance of notions among the academics, who isolate themselves from the real world, and thus are ignorant of it (even to this day), having produced nothing to progress the science in this field. The steadfast insistence on isolation, permits "progress" in their darkness, which in reality is pathetic regress.



Examples of their abdication:

- *Theory should be divorced from implementation concerns*
- *The server is a black box, physical, to be ignored*
- *The Transaction is irrelevant, a server concern*, etc.

Example of their false Straw Man notions:

- *Record Filing Systems marketed as the "Relational Model"*
- *The RM is broken*
- *SQL is broken*
- *A Transaction is anything between* `BEGIN` *and* `COMMIT`
- *Text only, visual modelling does not exist*
- *Pigs do fly and so can I*, etc.

In fifty years, there has not been a single paper that articulates or progresses the *Relational Model*.

In contrast, there has been hundreds of papers promoting this filth, or "inventing" wheels that were perfected in the industry fifty years earlier, vociferously ignorant of such facts due to their hysterical isolation, and plodding along with "progressions" that have no completion; no resolution, only more speculation upon speculation.

In staggering contradiction to their insistence on ignorance of implementation matters, they program young minds to produce an implementation, and market it heavily, on the ground of "academic" authority. In recent times, their backward and ignorant "methods" have found a brave new level of catastrophic failure: the proliferation of freeware. Eagerly embraced by the uneducated developers of today.

Due to their denial of the reality that the Stonebraker "method" did not work, and inability to determine that it is not science but fantasy, they repeat the same thing over again, expecting different results.

Oracle deserves a special mention. It sits squarely in this category:

- un-architected massive suite of programs masquerading as a "server"
- MV-non-CC with a bolted-on Lock Manager to make it work a little
- Not SQL compliant
- Cannot support ACID
- Fraudulent declarations

Much like MicroSoft in the last forty years, it is a terrible product, but with fantastic marketing.

For an understanding of architectural issues, the difference between program suites (freeware & Oracle) vs genuine commercial servers, refer to **Oracle vs Sybase**.

1 Not TCP/IP of today, but Terminal Control Program within CICS, that controlled the time-shared execution of multiple programs, each connected to a terminal. It was often called *Transaction Control Program* because Transactions were central; and a program executed just one Transaction: thus it resolved Transaction contention issues. At best, such programs were re-entrant: one pure code program used by multiple Terminals/users. For more detail, and a short trip into history, visit **History of IBM Mainframes**.

2 Historically, the initial and most reliable Transaction Processing method, well established, although the term ACID was not used. During the RDBMS wars, the benchmarks were fraudulent, due to vendors cheating on the definition of *Transaction*. The Transaction Processing Council standardised Transactions and coined the term ACID for the strict definition of *Transaction*.

OLTP Context vs MV-non-CC Isolation

SOFTWARE GEMS

OLTP Context

This illustrates elements relevant to Online Transaction Processing, and compares the emerging "RDBMS" freeware that is defined and heavily promoted by academia vs the established commercial RDBMS platforms, characterised by MV-non-CC vs Ordinary Locking. The scope is limited to the relevant issues herein, and related elements only (for the sake of completion), it is not a full comparison of those elements, nor an overall comparison. MySQL although freeware, and generally in the same category, has a paid engine, which is a far better implementation than PusGres.

Offline Version Processing (Single User, Schizophrenic)
- Isolation (contrived, tiled cell)
  - Nothing Shared
  - No sociability concerns
  - No concept of OLTP; ACID
- ACID not possible
- No Concurrency Control

Actual "Database" (RFS)

After-the-Fact Resolution
- Massive Churn
- More Versions

Commit
- Resolution of versions; fantasies
- "Serialisation Anomalies" = Fail

Additional Application Locks; Predicate Locks
- Proves "MVCC" is failure
- Dangerous (outside server); not SQL verbs

Clean Up
- Massive storage overhead
- Garbage collection
- Vacuum clean
- Shrink/Expand

- No Logical, 100% Physical
- No Normalisation
- Modelled & documented using
  - Anti-Relational ERD
- No Security (wide open)
- No Database Consistency
- OLAP not possible

Client Application (Multiple)

Private Version of "Database"
- Multiple x Users

"Transaction Isolation"
- All verbs, incl `SELECT`
- ? Transaction Consistency
- "Constraints" in app code

- Lost Updates (oblivious)
- No concept of currency: rows are versioned at retrieval, **not** the row TimeStamp

Online Transaction Processing
- Shared Resources
  - Social Behaviour (Rules)
  - Lock Manager (Enforcer, Concurrency Control)
  - Minimal contention (Various)
- ACID Context (Client & Server)
  - Synchronisation Failure; "Serialisation Anomalies", not possible

Shared Database (Single)
- Single version of Fact
- Server ACID
- Transaction context

Optimistic Locking
- High Concurrency
- No Lost Updates
- No Lost Currency

Client Application (Multiple)
- Retrieve
- Validate
- Exec Transaction

ACID Transaction
- Transaction ACID
- Invisible (Isolation)

Transaction
- Validate
- Execute

- Fully Relational (Logical)
- Fully constrained
  - DKNF per Codd Intent
- Defined as Predicates
- Modelled & documented in
  - IDEF1X/R, Logic Map
- Secured (No direct writes)
- OLAP is pedestrian

Dashed line means fantasy. *There is a sucker born every minute.*

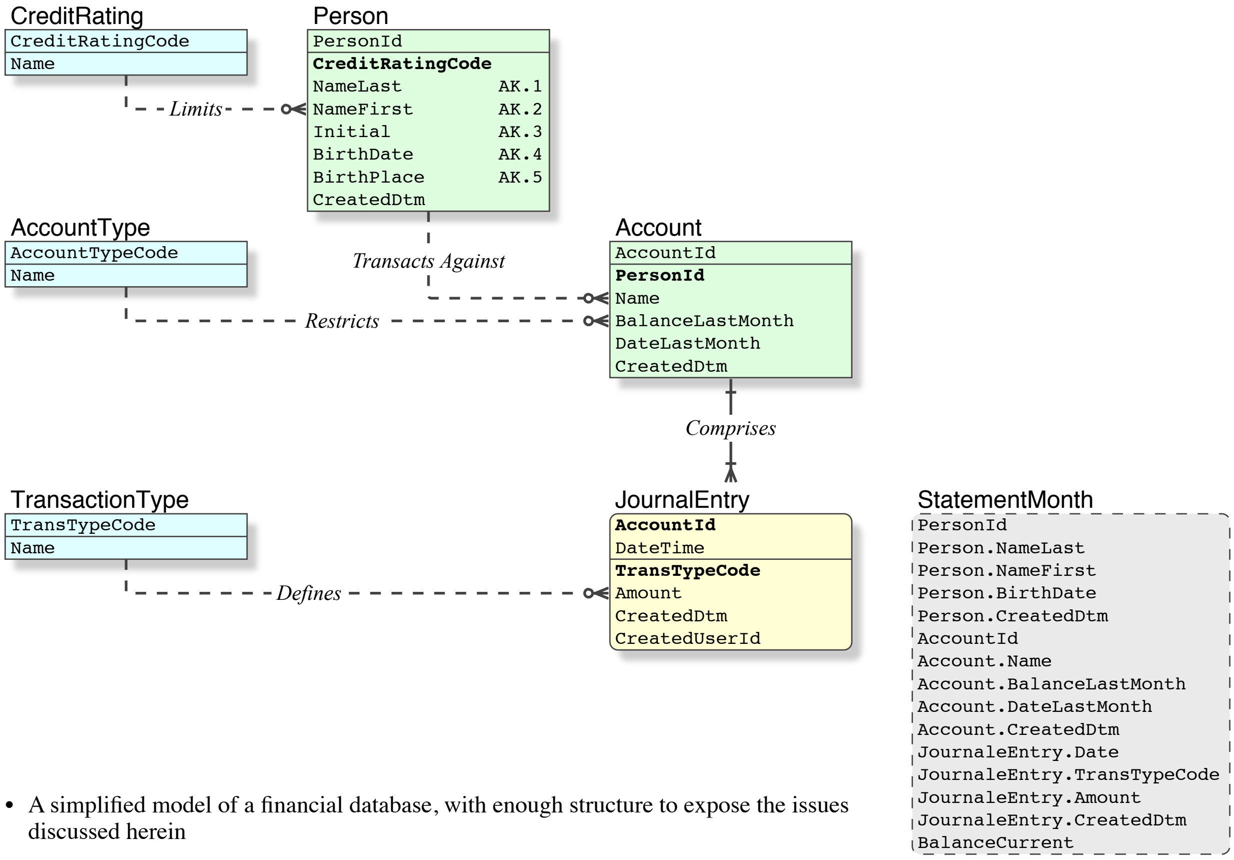
Transaction Consistency & Durability

This illustrates and allows comparison of the well-known Transaction Consistency problems (as distinct from Database Consistency), and their solutions, both well-known since 1965 (pre-Relational) and the 1980's (Relational). Due to the abject abdication of their role as scientists for the industry (insistent divorce from reality; from implementation concerns; etc), the academics are clueless, and are exposed to these problems only after implementation (yet another contradiction) of their speculations, fifty years after they were solved in reality. They are still speculating about the results of their speculations, with no solutions (either knowledge or more speculations) in sight. It also shows the contrast between SQL-compliant servers and non-compliant pretend "sql" program suites (pretend "servers").

	OLTP Consideration	Level	Pre-Relational Platform [1] Commercial SQL Platform	SQL Method	"SQL" Program Suite
	ACID Properties		Full ACID	OLTP Standard	ACID Not possible
	Reduction of [Implicit] Lock Contention		Standard	OLTP Standard	Contention Guaranteed
	Statement Integrity		`READ_COMMITTED`		Stale data [2]
Four Well-Known Issues	1 Phantoms; Anomalies	Statement Integrity		`BEGIN TRAN REPEATABLE_READ`	
Four Well-Known Issues		ResultSet (multi-Statement) Integrity		`BEGIN TRAN SERIALIZABLE`	
Four Well-Known Issues	2 Lost Updates, not Durable		Optimistic Locking		Unknown, Guaranteed [2]
Four Well-Known Issues	3 Lost Currency, not Consistent		Optimistic Locking		Unknown, Guaranteed
Four Well-Known Issues	4 Deadlocks		OLTP Standard/ Access Sequence		Guaranteed [4]
MV-non-CC	Synchronisation Failures; "Serialisation Anomalies" [5]		Not possible		Guaranteed [5]
MV-non-CC	"Pessimistic Locking" [6]		Prohibited		Guaranteed [6]
MV-non-CC	Lock Contention (Explicit, by Application) [7]		Not possible		Guaranteed [7, 8]

1 SQL Verbs are given. In pre-Relational Platforms, the access method & verbs were proprietary.
2 The acceptance of working with stale data is stupefying. More precisely, it is ignorance of the fact that the version, once obtained, is obsolete, and offline.
4 Not supposed to happen under MV-non-CC ... but it does (a) `serialzation_failure`, and (b) due to manual locking, which is required for Concurrency Control.
5 A special & unique feature of PoopGres, that other MV-non-CC program suites do not suffer, meaning that the limit of permutations of the fantasies has been reached. This is the result of (a) ignorance: that consistency regards contention resolution over shared resources (server design), and (b) stupidity: taking `SERIALIZABLE` as a directive regarding processing methods in the "server", rather than understanding it for what it is: a concept, as in *Transactions **appear** to be serialised*. Thus the fantasy, the hardened notion of "Transaction Isolation", in substitution for server design. It is refusal to understand ACID Properties, to fantasise instead.
6 The MV-non-CC groupies recite a mantra about Ordinary Locking, to make it look terrible, which is false because it is prohibited, it is pure Straw Man. In reality, it is the groupies that perform "pessimistic locking", and for uncontrolled durations, when they use manual locking, due to ignorance and lack of structure.
7 A Lock Manager is anathema in MV-non-CC ... but the failures are so bad that a primitive Lock Manager has been added. Apps that need Concurrency Control must use *manual, non-SQL* lock requests, which creates a new level of error (uncontrolled duration; deadlocks; etc), which is fatal (outside "server" control & resolution).
8 The manual Locking in **MySQL** uses named application locks, thus it is not vulnerable to interference by app code, or to deadlocks. If they occur, it is in the app.

Example Database[Bank Account]



- A simplified model of a financial database, with enough structure to expose the issues discussed herein

IDEF1X Notation

Lost Update

There are two major concurrency problems that are caused by the delay between obtaining a value from the database (painting it on the GUI), and the execution of the Transaction (either a proper stored proc in the server or a string of SQL in the app). These are problems that cannot be *expected to be* handled by an ACID-compliant server, or by an OLTP server, because the location of the problem is in app code, not in server facility. (The same as deadlocks: there are no deadlocks in the distribution, all deadlocks are *written* in the application.)

- The **Lost Update** problem, although lesser-known, is easier for novices to understand, thus it is given first.
- It is an ordinary consequence of that uncontrolled duration, but this problem is deadly to Data Integrity or Transaction Durability, not to OLTP proper. That is, the "transaction" fails the Durable property of ACID.
- There are many flavours or nuances to the Lost Update, this page illustrates one such species. Once it is understood, the other species can be appreciated, as being of the same genus.

User 1

Person has a Credit Rating **A** which allows **$10,000** overdraft
Person fails to keep agreements re covering their overdraft

Time

Credit Manager changes Credit Rating **D**; **$0** overdraft

T1

```
SELECT  @CreditRatingCode = CreditRatingCode,
        @BirthPlace       = BirthPlace
    FROM Person
    WHERE PersonId = @PersonId
```

User interaction, uncontrolled duration

User 2

Person has a balance of **$5,000 DR** (overdrawn)
Person phones bank clerk
Checks that he still has Credit Rating **A**
Changes his address

T2

```
SELECT  @CreditRatingCode = CreditRatingCode,
        @BirthPlace       = BirthPlace
    FROM Person
    WHERE PersonId = @PersonId
```

Uncontrolled duration,
No locks held,
Data Currency ignored

User interaction, uncontrolled duration

T3

```
BEGIN TRAN TweedleDumb
UPDATE Person SET
    CreditRatingCode ="D",
    BirthPlace       = @BirthPlace,
    WHERE PersonId   = @PersonId
COMMIT TRAN TweedleDumb
```

Succeeded, Credit Rating **D**

T4

```
BEGIN TRAN TweedleDumber
UPDATE Person SET
    CreditRatingCode = @CreditRatingCode,
    BirthPlace       = "Timbuktu",
    WHERE PersonId   = @PersonId
COMMIT TRAN Dumb_Two
```

Controlled duration,
Locks held,
Data Currency lost.
Separation of Validate &
Execute blocks (structure) is absent.

Succeeded, Credit Rating **A**
Lost Update: User 1 Credit Rating **D**

User 3

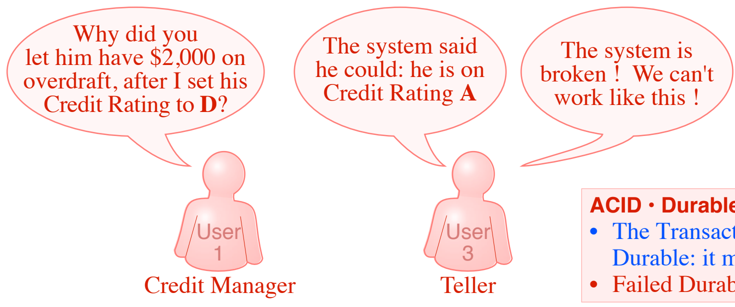
Person attempts to withdraw **$2,000** at the bank

T5

```
EXEC Dumb_Withdraw_tr
  @PersonId,
  @AccountId,
  @Amount = -2000
```

```
BEGIN TRAN Dumb_Withdraw
SELECT  @CreditRatingCode = CreditRatingCode,
        @Balance = BalanceLastMonth + (
        SELECT SUM(Amount)
            FROM JournalEntry
            WHERE AccountId = a.AccountId
            AND   DateTime >  a.DateLastMonth
            )
    FROM Person p,
         Account a
    WHERE p.PersonId  = @PersonId
    AND   p.PersonId  = a.PersonId
    AND   a.AccountId = @AccountId
SET @AmountLimit = CASE @CreditRatingCode
    WHEN "A" THEN @Balance + 10000
    WHEN "B" THEN @Balance + 1000
    ELSE          @Balance
    END
IF @Amount <= @AmountLimit
    BEGIN
    INSERT JournalEntry VALUES (
        @AccountId,
        GETDATE(),
        CASE WHEN @Amount < 0 THEN "DR"
             ELSE "CR" END,
        @Amount,
        GETDATE(),
        SUSER_ID()
        )
    COMMIT TRAN Dumb_Withdraw
    RETURN 0 -- Transaction Accepted
    END
ELSE
    BEGIN
    ROLLBACK TRAN Dumb_Withdraw
    RETURN 1 -- Transaction Rejected
    END
```

Succeeded, due to Credit Rating **A**, the Lost Update



ACID • Durable
- The Transaction must be Durable: it must survive
- Failed Durable

- Error checking & handling, which is mandatory for every verb, is excluded for brevity
- Time marks the moment of execution of the command or Transaction
- Green: resident in the Client
- Blue: resident in the server (Transaction stored proc)
- Pink: in the pink

Lost Currency

- The better-known problem is **Lost Currency**. It does not pertain to Data Integrity, but to Transaction Consistency. The causative problem is that delay between obtaining a value from the online shared database (painting it on the GUI), and the execution of the Transaction.
- Nevertheless, it needs to be understood logically (top-down), not merely physically (bottom-up), such that the problem is solved logically.

User 1

Person has a Credit Rating **A** which allows **$10,000** overdraft
Person fails to keep agreements re covering their overdraft

Time

Credit Manager changes Credit Rating **D**; **$0** overdraft

T1

```
SELECT  @CreditRatingCode = CreditRatingCode,
        @BirthPlace       = BirthPlace
    FROM Person
    WHERE PersonId = @PersonId
```

User interaction, uncontrolled duration

User 2

Person has a balance of **$5,000 DR** (overdrawn)
Person phones bank clerk

Checks that he still has Credit Rating **A**
Changes his address

T2

```
SELECT  @CreditRatingCode = CreditRatingCode,
        @BirthPlace       = BirthPlace
    FROM Person
    WHERE PersonId = @PersonId
```

Uncontrolled duration,
No locks held,
Data Currency ignored

User interaction, uncontrolled duration

T3

```
BEGIN TRAN TweedleDumb
UPDATE Person SET
    CreditRatingCode ="D",
    BirthPlace       = @BirthPlace,
    WHERE PersonId   = @PersonId
COMMIT TRAN TweedleDumb
```

Succeeded, Credit Rating **D**

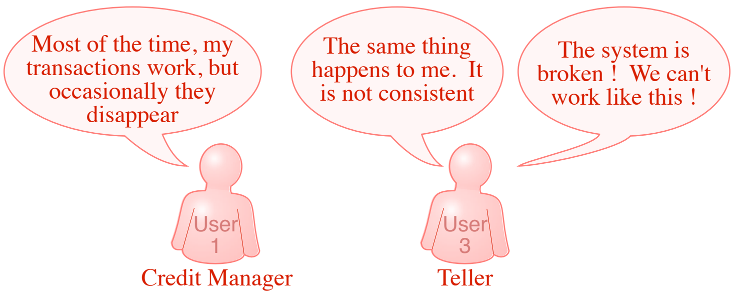
T4

```
BEGIN TRAN TweedleDumber
UPDATE Person SET
    CreditRatingCode = @CreditRatingCode,
    BirthPlace       = "Timbuktu",
    WHERE PersonId   = @PersonId
COMMIT TRAN Dumb_Two
```

Controlled duration,
Locks held,
Data Currency lost.
Separation of Validate &
Execute blocks (structure) is absent.

Succeeded, Credit Rating **A**
Lost Update: User 1 Credit Rating **D**

- Question: at Time[T3] and [T4], What is the **basis** for confidence, that the "transaction" is valid ?
- Answer: the hysterical notion that, in that uncontrolled duration between retrieval and execution, the online shared database **has not changed**.
- Again the schizophrenic mindset, the client-version-centric mindset, in denial of the reality of an online shared database, of the single version of the truth.



ACID · Consistent
- The Transaction must start, and leave, the database in a Consistent state
- The Transaction must be Consistent
- Failed Consistent

Naïve "Solution"

Unfortunately, the **Lost Currency** problem is only understood by novices when the naïve "solution" is examined, thus it is given here. The above relies on the notion that the data has not changed between retrieval and update: since that is false, they 'ensure' that it does not change by locking it.

User 1

Person has a Credit Rating **A** which allows **$10,000** overdraft
Person fails to keep agreements re covering their overdraft

Credit Manager changes Credit Rating **D**; **$0** overdraft

T1

```
BEGIN TRAN TweedleDumb
SELECT  @CreditRatingCode = CreditRatingCode,
        @BirthPlace       = BirthPlace
    FROM Person [HOLDLOCK]
    WHERE PersonId = @PersonId
```

User interaction, uncontrolled duration:
  coffee
  phone call
  toilet break

User 2

Person has a balance of **$5,000 DR** (overdrawn)
Person phones bank clerk

Checks that they still have Credit Rating **A**
Changes their address

T2

```
BEGIN TRAN TweedleDumber
SELECT  @CreditRatingCode = CreditRatingCode,
        @BirthPlace       = BirthPlace
    FROM Person [HOLDLOCK]
    WHERE PersonId = @PersonId
```

Uncontrolled duration,
***Locks held**,*
*Data Currency **forced***

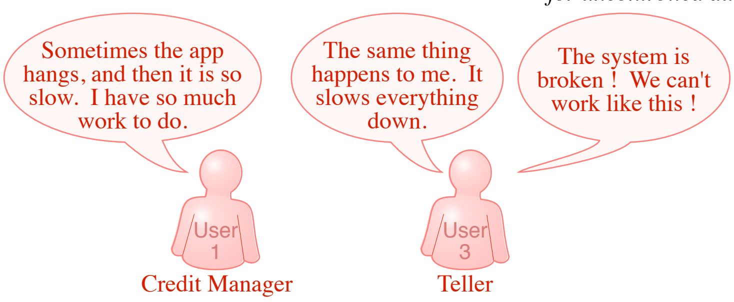
Locked out, suspended until [T3]

*Other users **Locked Out***
for uncontrolled duration

T3

```
-- Continue TRAN TweedleDumb
UPDATE Person SET
    CreditRatingCode ="D",
    BirthPlace       = @BirthPlace,
    WHERE PersonId   = @PersonId
COMMIT TRAN TweedleDumb
```

Succeeded, Credit Rating **D**

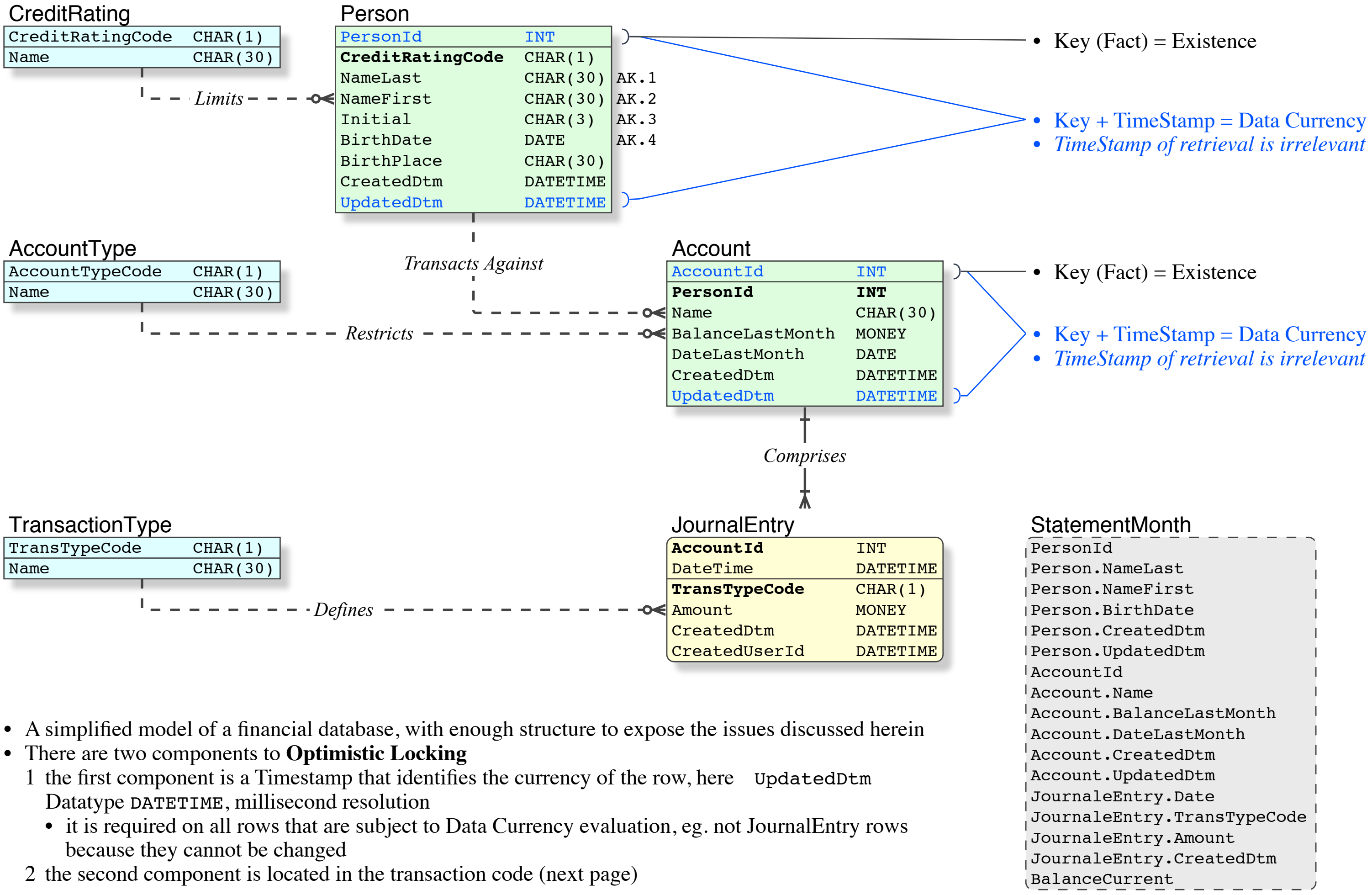


OLTP
- Typically uneducated developers and schizophrenics (Stonebraker cultists: denial of reality) lock the row upon retrieval: that guarantees a lock storm, and for uncontrolled durations
- That is fatal in an OLTP environment, thus it is prohibited
- The cure is worse than the disease, and imposed on all users
- Failed OLTP
- Insanity is not an alternative to sanity.

ACID · Atomic
- All changes to the database are performed as if they are a single operation
- The operation is all or nothing, indivisible
- The "transaction" is spread across the network, uncontrolled, not a single operation
- Failed Atomic

The Real "Pessimistic Locking"
1. The groupies label their MV-non-CC fantasy "optimistic", in the fervent hope that the label produces the effect (the maintenance of offline versions is definitely not optimistic)
2. They label the OLTP Method (which as evidenced, they do not understand), "pessimistic locking", as the perceived opposite of their "optimistic" label. It is a Straw Man
3. However, because their "MVCC" fantasy has no Concurrency Control whatsoever, and thus it does not work until a multi-layered Lock Manager is **added**, and used (including manual locking), in reality, it is their anti-method that employs "pessimistic locking" (lock on retrieval, shown here), and it guarantees the consequent problems.

OLTP Database[Bank Account]



- A simplified model of a financial database, with enough structure to expose the issues discussed herein
- There are two components to **Optimistic Locking**
  1. the first component is a Timestamp that identifies the currency of the row, here `UpdatedDtm`
     Datatype `DATETIME`, millisecond resolution
     - it is required on all rows that are subject to Data Currency evaluation, eg. not JournalEntry rows because they cannot be changed
  2. the second component is located in the transaction code (next page)

IDEF1X Notation

Optimistic Locking

Online Transaction Processing is a mindset, a set of rules for application development that recognises, and minimises, contention on shared resources. **Optimistic Locking** is a Method that eliminates two major concurrency problems: **Lost Currency**; and **Lost Update** (explained, previous pages). The solution to the two failed examples (previous pages) is given here. (The OLTP context & Transaction template are explained separately.)

User 1

Person has a Credit Rating **A** which allows **$10,000** overdraft
Person fails to keep agreements re covering their overdraft
Credit Manager changes Credit Rating to **D**; **$0** overdraft

Time

T1

```
SELECT  @CreditRatingCode = CreditRatingCode,
        @BirthPlace       = BirthPlace,
        @UpdatedDtm       = UpdatedDtm --[T0]
    FROM Person
    WHERE PersonId = @PersonId
```

T2

User interaction, uncontrolled duration

T3

```
EXEC PersonUpdate_tr
  @PersonId,
  @UpdatedDtm, --[T0]
  @CreditRatingCode = "D",
  @BirthPlace = BirthpPlace
```

T4

```
-- Validate
SET TRANSACTION ISOLATION LEVEL 1
SELECT  1
    FROM Person
    WHERE PersonId   = @PersonId
    AND   UpdatedDtm = @UpdatedDtm --[T0]
IF @@ROWCOUNT != 1
    RETURN 20002
```

```
-- Execute
BEGIN TRAN PersonUpdate
SET TRANSACTION ISOLATION LEVEL 3
SELECT  1
    FROM Person
    WHERE PersonId   = @PersonId
    AND   UpdatedDtm = @UpdatedDtm --[T0]
IF @@ROWCOUNT != 1
    BEGIN
    ROLLBACK TRAN PersonUpdate
    RETURN 20002
    END
UPDATE  Person SET
        CreditRatingCode = @CreditRatingCode,
        BirthPlace       = @BirthPlace,
        UpdatedDtm       = GETDATE() --[T3]
    WHERE PersonId   = @PersonId
    AND   UpdatedDtm = @UpdatedDtm
IF @@ROWCOUNT != 1
    BEGIN
    ROLLBACK TRAN PersonUpdate
    RETURN 20001
    END
COMMIT TRAN PersonUpdate
RETURN 0
```

Succeeded, Credit Rating **D**

User 2

Person has a balance of **$5,000 DR** (overdrawn)
Person phones bank clerk
Checks that he still has Credit Rating **A**
Changes his address

```
SELECT  @CreditRatingCode = CreditRatingCode,
        @BirthPlace       = BirthPlace,
        @UpdatedDtm       = UpdatedDtm --[T0]
    FROM Person
    WHERE PersonId = @PersonId
```

Uncontrolled duration,
No locks held,
Data Currency set

User interaction, uncontrolled duration

```
EXEC PersonUpdate_tr
  @PersonId,
  @UpdatedDtm, -- [T0]
  @CreditRatingCode = CreditRatingCode,
  @BirthPlace       = "Timbuktu"
```

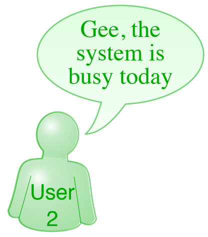
```
-- Validate
SET TRANSACTION ISOLATION LEVEL 1
SELECT  1
    FROM Person
    WHERE PersonId   = @PersonId
    AND   UpdatedDtm = @UpdatedDtm --[T0]
IF @@ROWCOUNT != 1
    RETURN 20002
```

Controlled duration,
No locks held
Data Currency validated

Failed, due to `@UpdatedDtm[T0] != UpdatedDtm[T3]`

Lost Update prevented

Likewise, Lost Currency prevented



User 3

Person attempts to withdraw **$2,000** at bank

T5

```
EXEC Withdraw_tr
  @PersonId,
  @AccountId
  @UpdatedDtm, --[T3]
  @Amount = -2000
```

```
-- Validate
SET TRANSACTION ISOLATION LEVEL 1
SELECT  1
    FROM Person  p,
         Account a
    WHERE p.PersonId   = @PersonId
    AND   p.UpdatedDtm = @UpdatedDtm --[T3]
    AND   p.PersonId   = a.PersonId
    AND   a.AccountId  = @AccountId
IF @@ROWCOUNT != 1
    RETURN 20002
```

```
-- Execute
```

Rejected, Credit Rating **D**

```
-- Execute
BEGIN TRAN Withdraw
SET TRANSACTION ISOLATION LEVEL 3
SELECT  1
    FROM Person  p,
         Account a
    WHERE p.PersonId   = @PersonId
    AND   p.UpdatedDtm = @UpdatedDtm --[T3]
    AND   p.PersonId   = a.PersonId
    AND   a.AccountId  = @AccountId
IF @@ROWCOUNT != 1
    BEGIN
    ROLLBACK TRAN Withdraw
    RETURN 20002
    END
SELECT  @CreditRatingCode = CreditRatingCode,
        @Balance = BalanceLastMonth + (
        SELECT SUM(Amount)
            FROM JournalEntry
            WHERE AccountId = a.AccountId
            AND   DateTime >  a.DateLastMonth
        )
    FROM Person  p,
         Account a
    WHERE p.PersonId   = @PersonId
    AND   p.UpdatedDtm = @UpdatedDtm --[T3]
    AND   p.PersonId   = a.PersonId
    AND   a.AccountId  = @AccountId
SET @AmountLimit = CASE @CreditRatingCode
    WHEN "A" THEN @Balance + 10000
    WHEN "B" THEN @Balance + 1000
    ELSE          @Balance
    END
IF @Amount <= @AmountLimit
    BEGIN
    INSERT JournalEntry VALUES (
        @AccountId,
        GETDATE(),
        CASE WHEN @Amount < 0 THEN "DR"
             ELSE "CR" END,
        @Amount,
        GETDATE(),
        SUSER_ID()
        )
    COMMIT TRAN Withdraw
    RETURN 0 -- Transaction Accepted
    END
ELSE
    BEGIN
    ROLLBACK TRAN Withdraw
    RETURN 1 -- Transaction Rejected
    END
```

Indicates change to Person tree

Controlled duration,
Locks held,
Rollback required,
Data Currency validated

ACID · Durable
- The Transaction must be Durable: it must survive
- The platform supplies system durability.

ACID · Atomic
- All changes to the database are performed as if they are a single operation

OLTP · Atomic, Controlled
- Contiguous code block
- No user interaction within a Transaction.

Large Transaction
- This Method can be optimised further; storing the Timestamp and validating every row within a multiple-row transaction is not always necessary.
- The Timestamp of the top-most row in an hierarchy can be used to identify the status of the tree.
- Here the `Person.UpdatedDtm` is used to indicate any change to the Person's `Person`; `Account`; or `JournalEntry` rows.

```
Error 20001, '%1!: Encountered error %2!.  %3!'
Error 20002, '%1!: Data in %2! has changed between initial data retrieval and transaction submission.'
Error 20004, '%1!: A transaction has been opened by the caller (but is declared NOT to be open).'
Error 20005, '%1!: A transaction has NOT been opened by the caller (but is declared to be open).'
Error 20006, '%1!: is an utility transaction, it must be called from within a open transaction.'
```