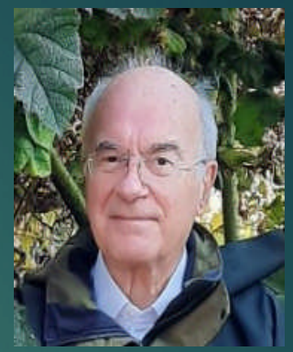
Data: Evolution and Durability

MALCOLM CROWE, FRITZ LAUX

IARIA 2022

Malcolm Crowe

University of the West of Scotland
Email: malcolm.crowe@uws.ac.uk



- Malcolm Crowe is an Emeritus Professor at the University of the West of Scotland, where he worked from 1972 (when it was Paisley College of Technology) until 2018.
- He gained a D.Phil. in Mathematics at the University of Oxford in 1979.
- He was appointed head of the Department of Computing in 1985. His funded research projects before 2001 were on Programming Languages and Cooperative Work.
- Since 2001 he has worked steadily on PyrrhoDBMS to explore optimistic technologies for relational databases and this work led to involvement in DBTech, and a series of papers and other contributions at IARIA conferences with Fritz Laux, Martti Laiho, and others.
- Prof. Crowe has recently been appointed an IARIA Fellow.

Prof. Dr. Fritz Laux

(Retired), Reutlingen University
Email: fritz.laux@reutlingen-university.de

- Prof. Dr. Fritz Laux was professor (now emeritus) for Database and Information Systems at Reutlingen University from 1986 - 2015. He holds an MSc (Diplom) and PhD (Dr. rer. nat.) in Mathematics.
- His current research interests include
  - Information modeling and data integration
  - Transaction management and optimistic concurrency control
  - Business intelligence and knowledge discovery
- He contributed papers to DBKDA and PATTERNS conferences that received DBKDA 2009 and DBKDA 2010 Best Paper Awards. He is a panellist, keynote speaker, and member of the DBKDA advisory board.
- Prof. Laux is a founding member of DBTech.net (http://www.dbtechnet.org/), an initiative of European universities and IT-companies to set up a transnational collaboration scheme for Database teaching. Together with colleagues from 5 European countries he has conducted projects supported by the European Union on state-of-the-art database teaching.
- He is a member of the ACM and the German Computer Society (Gesellschaft für Informatik).

Evolution and Durability

- At first sight these look like "complementary" notions
  - Like position vs momentum, truth vs clarity
- For the best sorts of data, both are needed
  - What is the value? What was it before? Why changed?
    - Patient records, bank accounts, scientific results, guidelines
  - Copies, models and hearsay are likely to be wrong
    - Insist on correctness rather than availability
- This talk is about new approach to DBMS implementation
  - Taking account of changes since 1970s
  - Proof of Concept in StrongDBMS and PyrrhoDB (in progress)
- Full references in notes pages of these slides and at end

- DBMS should evolve

DBMS need to evolve too

- Durable storage is for what we want to keep
  - Don't use it for intermediate results or indexes
  - Instead append to the durable transaction log
- Make better use of the Internet service
  - Identify data ownership, provenance, auditing
  - Derive results from sources, not clones/copies
- Data is more durable than systems, devices
  - Legacy vs. history, alter vs. replace
- Better handling of remote data, not ETL
  - SQL needs to evolve

Better standards for DBMS

- Validate transaction serialization
- Support more of SQL standard (ISO 9075)
  - Including side effects in atomicity rule
    - Constraints, cascades, triggers
- Definer's role for each step of execution
  - A novel proposal to fix SQL's security model
- Generalize the data type system
- Support metadata directly in SQL
  - For all database objects including subtypes
    - Example: Specify inverse and monotonic functions
- Allow remote access to databases in SQL
  - Include a remote table in transaction control
    - Serialized transactions

Serialized Transactions

- The goal of any DBMS
  - Should be to serialize transactions
  - Many users making changes
    - Could lead to chaos
  - Transactional systems avoid this
    - cost of ~9% performance reported on some commercial systems
    - Alas: Business customers don't think this is worthwhile ☹
- Isolation levels defined in ISO standard
  - READ_UNCOMMITTED, READ_COMMITTED, REPEATABLE_READ, SERIALIZABLE
  - Textbooks say serializable is needed
  - But immediately settle for much less ☹
- A serialized transaction log (StrongDBMS, Pyrrho) ☺
  - Even better: Guarantees isolation by preventing conflicts

- Isolation and conflict

What is a conflict?

- Changes to the same database object
- For tables we fine granularity:
  - Report conflict if any columns read have been updated by another transaction
  - If only specific rows read, limit the above checks to these
- In 2021 PyrrhoDB demo with 50 clerks
  - Showed a high-concurrency version of TPC-C
  - The algorithm was re-implemented this year using two simple trees for columns and rows
    - Side effects

Side effects and atomicity

- Few DBMS implement this rule of SQL
- Consequential actions are part of transaction
- Cascades for DROP, DELETE, UPDATE constraints
  - DEFERRED actions should be done before transaction is committed
  - NO ACTION should be prohibited
- Side effects of evaluating constraints
- Anything done by triggers
- Recall that changes during a transaction are not visible to other users
  - But may throw exceptions that abort the transaction
- All become visible on COMMIT

- Definer's role

Definer's role in execution

- IBM DB2 and Oracle have this feature
  - It's important when allowing remote access
- Suppose the current role is D
  - D defines a procedure P
  - D grants P to role R
- User U with usage privilege on R can call P
  - Execution of P will have D's privileges
  - Similarly for a call within P
- Similarly for table contents and view definition
  - Have insert, delete privileges
  - Select and Update at table or column level
- However, all types belong to SYSTEM

Generalize the type system

- SQL's compatibility rules require equal precision and string length
  - Should allow to alter columns to greater length
  - Should allow to alter seconds precision etc
- SQL allows the definition of subtypes
  - Of user-defined types using UNDER
  - Should allow it for predefined types too
  - Should regard CHAR(5) as a subtype of CHAR
  - Should regard a user defined type as a subtype of its underlying type
- Where a user defined type is expected, a subtype can be assigned
  - This should be possible for general subtypes
- It should be possible to have subtypes of predefined types
  - And row types
- SQL already allows type predicates (OF) and create table of type

- Metadata support

Metadata

- Experimental in Pyrrho
  - Almost any DDL command can add or drop metadata
  - Currently 24 metadata ids, some with args
    - Most affect HTTP service or XML/JSON output
  - Some for updatable views etc (e.g. INVERTS)
    - If a view V uses a procedure for one or more columns, it will not be updatable unless there is an inverse function has been defined

- Big Live data

Big *Live* Data

- If your data originates in lots of databases
- You could copy the data centrally
  - Extract-Transform-Load/Big Data
- But, if it keeps changing this is not good
  - The durable record should be accessed now
  - And leave data where it is evolving (or curated)
- So, suppose our data is remote
  - A table's rows come from different databases
    - E.g. Sales or product data from different companies
- The available data is provided as a View
  - And accessible using HTTP and JSON

- Next: A derived table

A derived table

Derived = not actually stored centrally

Columns from D's renamed and values probably transformed

CID	A	B	C	…			
D1							
D1							
D2							
D3							
D3							
D3							

(Contributors take responsibility for renaming columns and transforming data to suit us as their schemas will all be different)

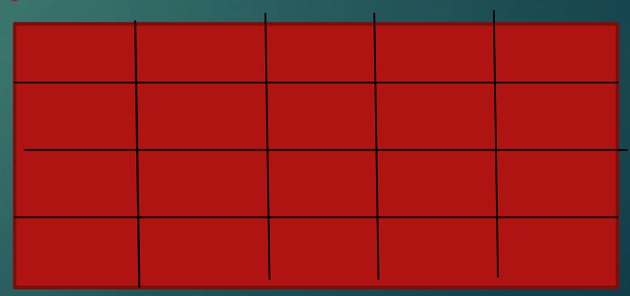
Next: Contributing DBMS

Defining a contribution

- Probably, each contributor creates a VIEW
  - Out of data from one or more actual tables

CREATE VIEW (A,B,C..) AS ….

A	B	C	…			



Can identify each contributor in the result view with a contributor id CID and maybe other information

- Next: The central view

Centrally we then have

- A row type CID,..,A,B,C,..
  - The local row contains remote data
- A local table T of contributor details, URLs

T:

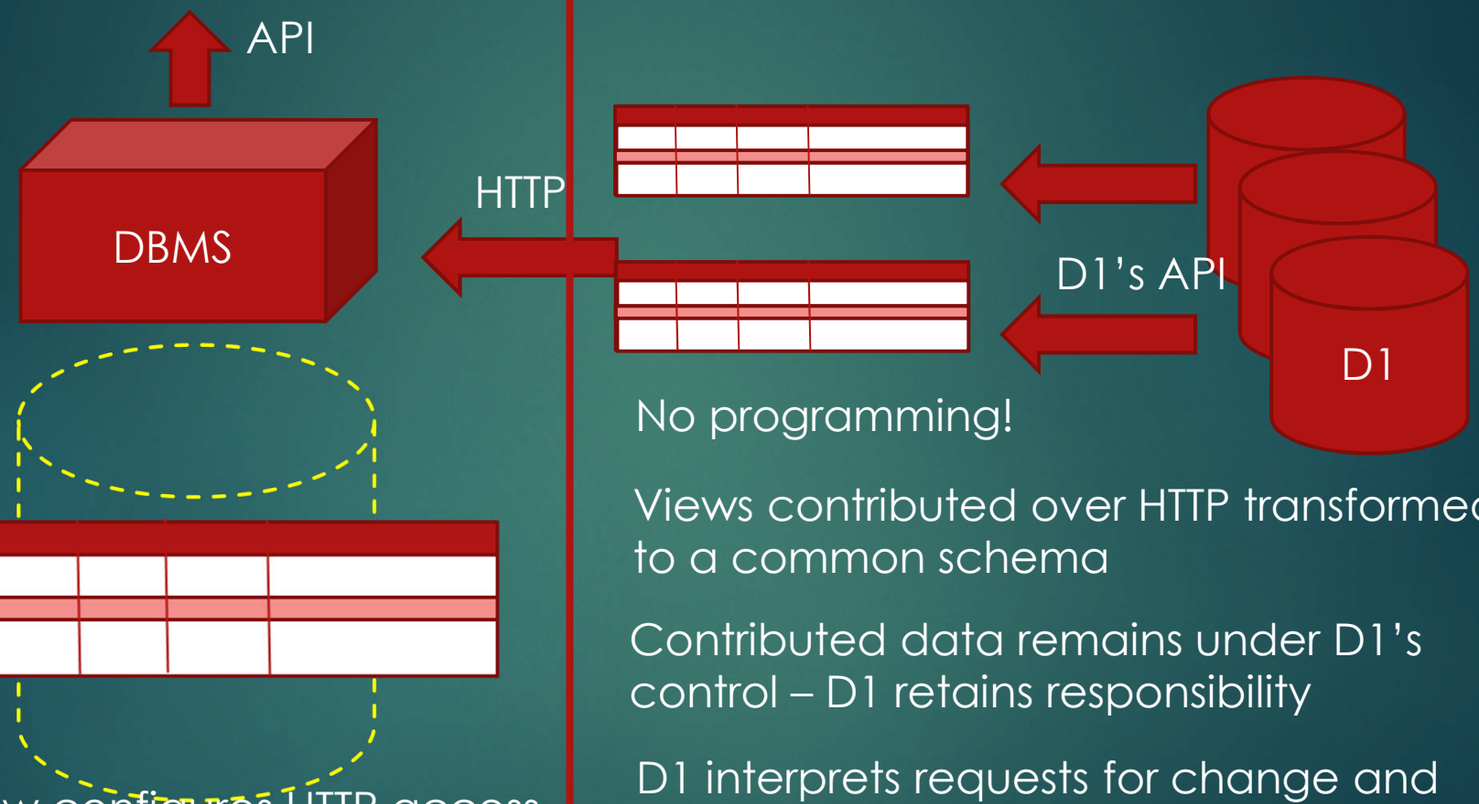
CID	…	URL
D1	…	URL for D1's data
D2	…	URL for D2's data
D3	…	URL for D3's data

CREATE VIEW V OF (CID..,A,B,C..) AS GET USING T

- OF clause gives V's row type (specifying column data types)
  - Includes all columns from T except the last (the URL)
  - The remaining columns specify the data from the remote view

- Next: Dividing responsibility

Division of responsibility



No programming!

Views contributed over HTTP transformed to a common schema

Contributed data remains under D1's control – D1 retains responsibility

D1 interprets requests for change and inverts the transformations if it can

View configures HTTP access

Change request sent to D1,..

- Next: View-mediated access

What happens with REST

- REST operations use standard formats
- For rows, we use JSON documents
- An item for each column of the row
- Why not add some extra columns for the Registers in that row?
- A Register for each occurrence of an aggregation function in the select list
  - We define how to represent a Register in JSON

- Next: an example

Extra Register fields

- The local and remote servers see the same value expression
  - So the registers are supplied in the left-to-right ordering
- As a Json document with the following items:
  - The string value accumulated by the function if any
  - The value of MAX, MIN, FIRST, LAST, ARRAY
  - A document containing numbered fields for a multiset value
  - The value of a typed SUM
  - The value of COUNT
  - The sum of squares (if required for standard deviation etc)

- Next: Example

Transactions and REST

- All data needs a single transaction master
  - Because of the two-army problem
- Transactions start from one database
  - Called the local database (i.e. local server)
  - There is no way to address a remote object directly
- Some fields may come from remote views
  - Possibly updatable via REST over HTTP1.1 (safe)
  - At most one remote update can be allowed
- When the local commit is called
  - Local database locked, validation performed
  - The single remote update is done via HTTP1.1
  - And then the local commit can complete/unlock

- Next: Object-Orientation

Conclusions

- This research provides new DBMS tools
  - Serialized transactions
- Big Live Data implementation
  - Providing better real-time owned behavior
  - Optimized for aggregations of remote views
- Versioned API for transaction-safe apps

- Next: Links

Links

Crowe, M. K., Matalonga, S.: Shareable Data Structures, on https://github.com/MalcolmCrowe/ShareableDataStructures

- includes source code for StrongDBMS, PyrrhoV7alpha and documentation

Crowe, M. K., Laux, F.: Implementing True Serializable Transactions, Tutorial, DBKDA 2021

- https://www.youtube.com/watch?v=t4h-zPBPtSw&t=39s
- https://www.iaria.org/conferences2021/filesDBKDA21/

- Version 6.3: https://pyrrhodb.uws.ac.uk
- 50 clerks demo: https://youtu.be/0YaU59LvgLs
- Pyrrho blog: https://pyrrhodb.blogspot.com

- Next: References

References

Crowe, M. K., Laux, F.: Reconsidering Optimistic Algorithms for Relational DBMS, DBKDA 2020

Crowe, M. K., Matalonga, S., Laiho, M: StrongDBMS, built from immutable components, DBKDA 2019

Crowe, M. K., Fyffe, C: Benchmarking StrongDBMS, Keynote speech, DBKDA 2019

Crowe, M. K., Laux, F.: DBMS Support for Big Live Data, DBKDA 2018

Crowe, M.K., Begg, C.E., Laux, F., Laiho, M: Data Validation for Big Live Data, DBKDA 2017

Krijnen, T., Meertens, G. L. T.: "Making B-Trees work for B". Amsterdam : Stichting Mathematisch Centrum, 1982, Technical Report IW 219/83