Data: Evolution and Durability

MALCOLM CROWE, FRITZ LAUX

SOFTNET 2022 CONGRESS

Malcolm Crowe

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

IARIA

- 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).

Plan of this presentation

- What Evolution and Durability mean
- What is needed:
  - Some changes to SQL
  - Simplification of the security model
  - Practical steps for Big Live Data
- Conclusions

Next: DATA EVOLUTION ..

IARIA

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

- Data evolves

IARIA

Evolving data

- Always the focus of Relational DBMS
  - Customer accounts, scientific results
  - Shared access and long-term durability
  - Standards development continues today
    - With a cost: durability, backward compatibility
- Trend to use universal types, time, locale
- Big Data focus on metadata and semantics
  - Databases need to include such aspects

- Some use cases

IARIA

Big data: serious use cases

- Raw scientific and administrative data
  - Carried on the public web, often real time
- DNA signatures of new Covid variants
- Data from tsunami observatories
- Treatment history of seriously ill patients
- Fluid flow computations
- Steel plates used in a tower block, ship
- Available intensive care equipment
- A particular sensor in the Internet of Things

- We would like..

IARIA

A wish list for SQL support

- Search current data from named servers
- Search by metadata (RDF, provenance)
- Results include provenance and ownership
- Remote updates (if permitted) handled by owner
- Minimise data traffic, load on remote servers
- Allow for transformation during retrieval
  - With inverses for updates if permitted
- Changes securely transacted and durably recorded

- DBMS need to evolve

DBMS need to evolve too

- Durable storage is for what we want to keep
  - Don’t use it for intermediate results or indexes
  - Commits are added to the transaction log
    - Nothing else is ever written to durable storage
- 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
- Access data at its source: don’t use ETL

- SQL needs to evolve

Better standards for DBMS

On the next few slides we discuss the following ideas

- 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 help apply 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 remote tables in transaction control

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

- What are conflicts?

Managing transaction conflict

- Changes to the same database object
- For tables we have 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 (sec 4.41)
- 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
  - Condition handlers, exceptions
- 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

- Next: The Security model

IARIA

The SQL security model

- Most businesses use app-level security
  - Many have tried to implement roles
- SQL mandates Users and Roles
  - Many kinds of privileges on DB objects
    - But few suggestion on how to do this well
- We offer some suggestions here
- We assume operating system is secure
  - Authenticates users (DBMS shouldn't)
  - And secure communications over TCP

- What we would like

From user model to roles

- US Department of Defense Orange Book
  - Focus on user responsibility and security
- DBMS should focus on database objects
- Roles offer privileges on objects
  - And Users are allowed to use Roles
    - E.g. Access to all Sales or Finance tools, data
- Some suggestions:
  - User can use only one role at a time
    - Means that people can substitute for sick colleague
  - Auditing of all actions logs user and role
    - Facilitates investigation, remedies for bad actions
  - Avoid external routines: ensure DBMS in control
    - Use Definer's role

Definer's role

- Roles use different jargon and conventions
  - Naming of objects can depend on roles
- Focus on creators of database objects
  - Methods, tables, constraints, triggers
  - They will use conventions of their role
  - The finished object is then grantable
- Such code will work best in that role
  - Other staff might need to be given access
    - But surely not to all the underlying detail!

Standard implementation

- Evaluation of expressions uses roles
  - Object constraints and triggers
  - Invoked in background, use definer role
- The SQL standard has a context stack
  - New stack frames with correct privileges added on invocation, removed on return
- All data is passed in
  - Schema objects use their definer's role

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 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

- 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 transforms the value of a column, it will not be updatable unless there is an inverse transformation back to the base table's format

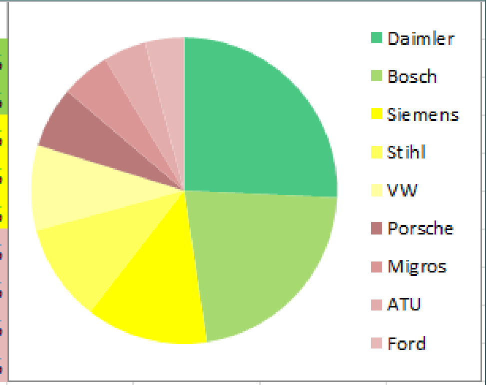
Big *Live* Data

- If your data originates in lots of databases
  - E.g. Sales or product data from subsidiary companies
- 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)
- The available data is provided as a View
  - And accessible using HTTP and JSON

Making big live data easier

- Today this needs detailed programming
- The following slides offer an SQL solution
- Define a VIEW for filtering specific data of interest
- Allow specific remote users some access to it
  - Maybe including updates for known users
- Then aggregations and filters do not need programming
  - Just write the SQL you want as if it was a local database
- Many examples in the Pyrrho v7 documentation

ABC	Kunde	Umsatz	kum. U.	kum. U. %
A	Daimler	20000	20000	26%
A	Bosch	17000	37000	48%
B	Siemens	10000	47000	60%
B	Stihl	8000	55000	71%
B	VW	7000	62000	80%
C	Porsche	5000	67000	86%
C	Migros	4000	71000	91%
C	ATU	3500	74500	96%
C	Ford	3200	77700	100%



- 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

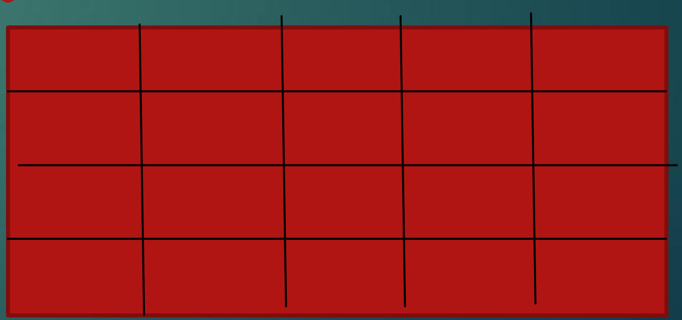
Defining a contribution

- 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

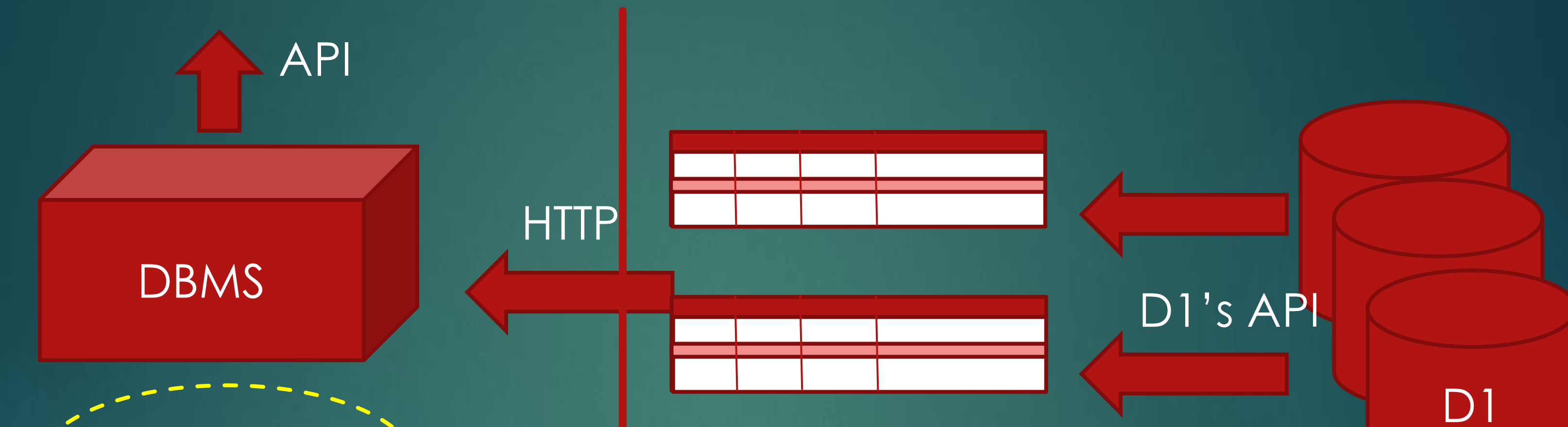
CREATE VIEW V OF (CID..,A,B,C..) AS GET USING T

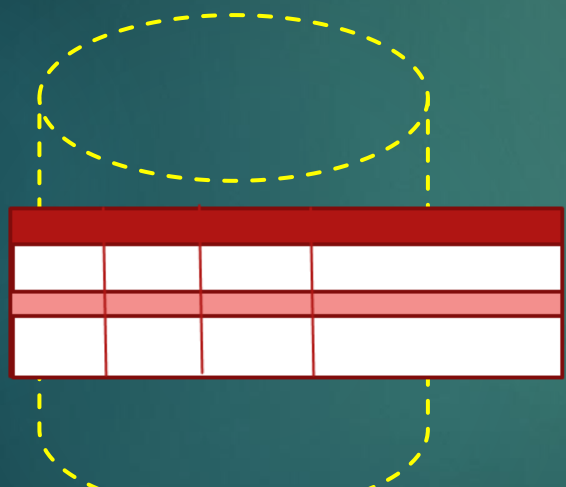
- OF clause gives V's row type (specifying column data types)
  - Includes all columns from T except the last (the URL)
  - The remaining columns specify the data from the remote view

- Next: Dividing responsibility

IARIA

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 transactions, use RFC7232 (ETags)
- 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
  - With a JSON representation

- Next: an example

A simple example

- Suppose we have a VIEW WW(E,F). Instead of select E,F we want

select sum(e)+char_length(f),f from ww group by f

- Simply send the query as is: Each database returns its answer
- The data from each has extra fields: The Registers for aggregates by group
- Unpacked and combined by Pyrrho

```
http://localhost:8180/DB/DB select (SUM(E)+CHAR_LENGTH(F)),F from t group by F
HTTP POST /DB/DB
select (SUM(E)+CHAR_LENGTH(F)),F from t group by F
Returning ETag: "23,-1,180"
--> 4 rows
Response ETag: 23,-1,180
```

```
SQL> select sum(e)+char_length(f),f  from ww group by f
|----|-----|
|Col0|F    |
|----|-----|
|11  |Ate  |
|9   |Five |
|8   |Four |
|11  |Sechs|
|9   |Six  |
|8   |Three|
|8   |Vier |
|----|-----|
SQL>
```

- More about registers

IARIA

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 as needed:
  - The string value accumulated by the function if any
  - The value of MAX, MIN, FIRST, LAST, ARRAY
  - A document containing counted values for a multiset value (can also be used for median, mode etc)
  - The value of a typed SUM
  - The value of COUNT
  - The sum of squares (if required for standard deviation etc)

- Transactions again

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

IARIA

Conclusions

- This research provides new DBMS tools
  - Serialized transactions, RESTViews etc
  - In PyrrhoDB v7.01 currently alpha
- Big Live Data implementation
  - Providing better real-time owned behavior
  - Optimized for aggregations of remote views
- Versioned API for transaction-safe apps
  - Schema verification (incl RESTView soon)

- Next: Links

IARIA

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

IARIA

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