DBMS Support for Big Live Data

MALCOLM CROWE AND FRITZ LAUX

Malcolm.Crowe@uws.ac.uk
Fritz.Laux@Reutlingen-University.de

Components proposed

- A syntax for virtual tables: "REST-Views"
  - With an optional table listing similar remote DBS
- A vendor-neutral HTTP transport for linking
  - Using simple SQL (minimising special features)
- Clever transformations for complex queries
  - Generated automatically from original view def
- Reversible transformations for alignment

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
  - Much better to read just what we need now
  - And leave data where it is being maintained
- So suppose our data is remote
  - A table's rows come from different databases
    - E.g. Sales or product data from different companies

Data is not owned by us

- Much of "Big Data" is randomly harvested
  - Schemaless, unstructured, for "exploration"
- And we didn't arrange it with anyone
  - So we have really no idea of semantics
- With GDPR there will be less such data
- Instead we should discuss with providers
- What data they are able/willing to share
- And how we can best make use of it
  - Subject to their restrictions on volume, intrusion

Such negotiations cost

- Once we have settled what we want
  - We don't want to keep going back
- Our DBMS should avoid this need
  - No programming or complex protocols
  - Just automatic transformation of views
- We have no detailed knowledge of data
  - So we just minimise what we get sent
  - By intelligently querying the remote DB
- So: they agree to supply us VIEWS
  - E.g.: We are government/UN/group HQ/admin

Use HTTP and Json

- Instead of proprietary DBMS connectors
- They give us a login ID to access the data
- And we give them a tiny Web server WS
  - Such interfaces are easy to write
- We POST SQL statements over HTTP/HTTPS
  - Providing the credentials they have given
- WS uses their DBMS connector to execute
- And send us the results in Json format
  - We are going to make this lightweight

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)

Contributing databases

- Contributors provide data in a given form
  - On request, using HTTP with REST/JSON format
- They probably don't have it in this form
  - So they create a VIEW with the right columns
  - Values probably requires some transformation
- Make it available with a given URL
  - With access permissions for our view
  - Possibly they might allow some updates

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	…			

Centrally we then have

- The row type CID,A,B,C,..
- The list of contributors with their URLs

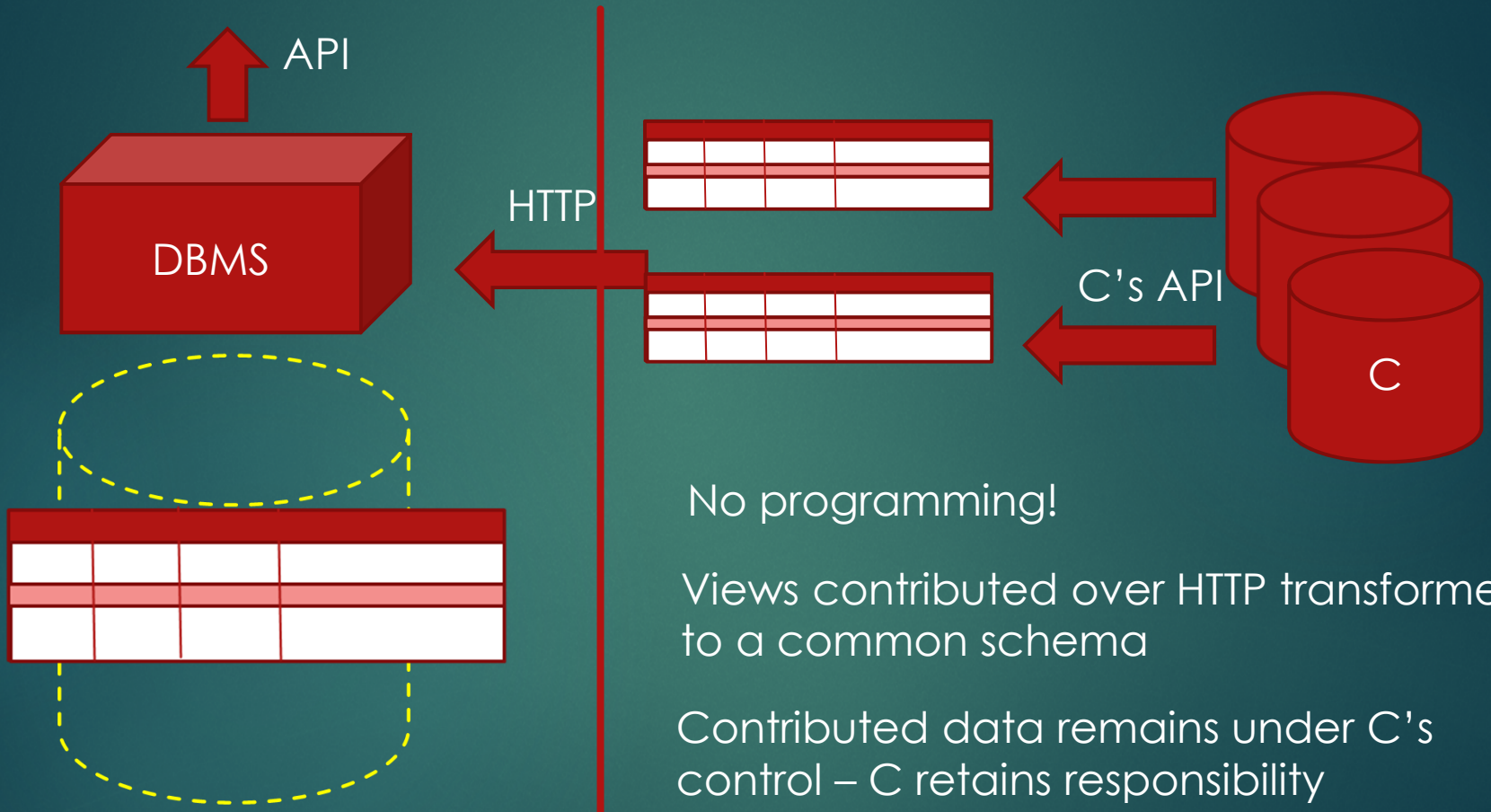
CREATE VIEW DT OF (CID..,A..,B..,C..) AS GET USING T

T:

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

- OF gives DT row type (with column data types)
  - All columns from T except the last (CID here)
  - The remaining columns specify the remote view

Division of responsibility



View configures HTTP access

Change request sent to C

No programming!

Views contributed over HTTP transformed to a common schema

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

C interprets requests for change and Inverts the transformations

Transforming the query

- As defined the view has a simple table form
  - But we don't want to get even 1MB of data
  - Only select required columns, apply filters
- Joins and aggregations get interesting
  - We can perform many aggregations remotely
  - So we only get a few rows (maybe just one)
- A query can join these with local data
  - And optimising such a join is a great idea
- Always leave getting data to after analysis

For example

- If W is defined as a join with remote data V
  - Aggregating V's data, GROUP BY a,b,..
- The grouping operation can be remote
  - Provided we also group by the joined columns
- View definitions, subqueries, joins
  - All lead to known matching columns, exprs
  - We can use these when optimising
- We will have some predefined views, joins
  - That consume data coming from the remote V

Query Rewriting 101

- SQL query is a recursive composite structure
- CursorSpecification
  - QueryExpression (union/intersect etc)
    - QuerySpecification (Select List)
      - TableExpression (Aggregation | Grouping)
        - Table | View | SubQuery
          - CursorSpecification
            - …
        - [redacted]
  - [redacted]
    - T
- Select items can contain query expressions
  - Filters (where conditions) can go anywhere