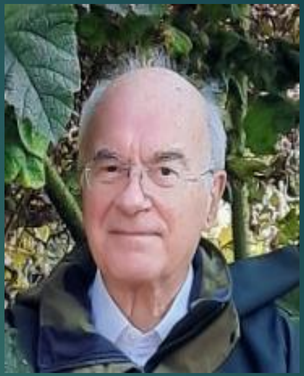
Graph Data Models

MALCOLM CROWE, FRITZ LAUX

DBKDA 2023

Malcolm Crowe

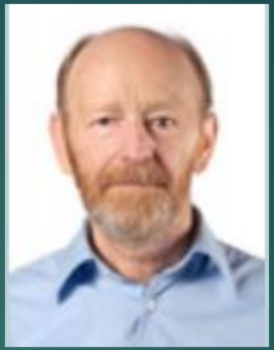
University of the West of Scotland
Email: malcolm.crowe@uws.ac.uk



- 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).

Plan of this presentation

- The Typed Graph Model TGM (review)
- TGM and relational data
  - Why a combined approach?
- Graph Modeling approach:
  - Creating a TGM by instances
  - And using MATCH to query its contents
- RDBMS version
  - Creating and modifying using SQL
- Current status and conclusions

The Typed Graph Model

- A typed graph schema is a tuple $TGS=(N_S,E_S,\varrho,T,\tau,C)$ where:
- N_S is the set of named (labeled) objects (nodes) n with properties of data type $t:=(l,d)\in T$, where l is the label and d the data type definition.
- E_S is the set of named (labeled) edges e with a structured property $p:=(l,d)\in T$, where l is the label and d the data type definition.
- ϱ is a function that associates each edge e to a pair of object sets (O,A), i. e., $\varrho(e):=(O_e,A_e)$ with $O_e, A_e \in \wp(N_S)$. O_e is called the tail and A_e is called the head of an edge e.
- τ is a function that assigns for each node n of an edge e a pair of positive integers (i_n,k_n), i. e., $\tau_e(n):=(i_n,k_n)$ with $i_n \in N_0$ and $k_n \in N$. The function τ defines the min-max multiplicity of an edge connection. If the min-value i_n is 0 then the connection is optional.
- C is a set *of integrity constraints*, which the graph database must obey.

- Why RDBMS support

IARIA

Reasons to add SQL support

- The SQL programming model is well known
- Most organisations have an RDBMS so it avoids having a separate product and support team
- SQL queries can process graph data
- Graph methods can be used for SQL data

- An example

An example: graph creation

```
CREATE
(Joe:Customer {"Name":'Joe Edwards',
Address:'10 Station Rd.'}),
(Joe)-[:Ordered {"Date":date'22/11/2002'} ]->
(Ord201:"Order")-[:Item {Qty: 5}]->
("16/50x100" : Woodscrew : Product),
(Ord201)-[:Item {Qty: 5}]->("Fiber 12cm" :
Wallplug: Product),
(Ord201)-[:Item {Qty: 1}]->("500ml" :
Rubberglue : Product)
```

Schema Implementation

- The TGM can be implemented in a relational DBMS as follows:
- Each node type and edge type defines a base table, whose rows are the node and edge instances
- There is a predefined primary key ID for both nodes and edges, which is an autokey
- The relationship of edges to nodes is as two predefined foreign keys LEAVING and ARRIVING in each edge table
- Node and edge properties are columns in the node and edge types
- We support subtypes for edge types

- A graph query

A graph query

```
MATCH (_)-[:Item {Qty:_Q}]-> (_Y:_T)
where Q>4
```

```
SQL> match (_)-[:Item {Qty:_Q}]->(_Y:_T) where Q>4
|-|-----------|----------|
|Q|Y          |T         |
|-|-----------|----------|
|5|16/8x100   |WOODSCREW|
|5|Fibre 12cm|WALLPLUG  |
|-|-----------|----------|
```

Graph definition

- If a graph is entered as in Neo4j by giving node and edge instances, the graph and edge types are incrementally inferred by the DBMS engine
- Nodes (..) and Edges (..)–[..]->(..) (..)<-[..]-(..) can be strung together, so a graph can be constructed by CREATE and a comma-separated list of instances
- Nodes and edges can be introduced id:label with properties in JSON notation

```
(Joe:Customer {Address:'10 Station Rd'})
```

- And similarly for edges
- Nodes can be later referenced using their ID

```
(Joe)
```

- The properties of a node or edge once defined can only be changed using SQL

- Graph creation again

An example graph creation

```
CREATE
(Joe:Customer {"Name":'Joe Edwards',
Address:'10 Station Rd.'}),
(Joe)-[:Ordered {"Date":date'22/11/2002'} ]->
(Ord201:"Order")-[:Item {Qty: 5}]->
("16/50x100" : Woodscrew : Product),
(Ord201)-[:Item {Qty: 5}]->("Fiber 12cm" :
Wallplug: Product),
(Ord201)-[:Item {Qty: 1}]->("500ml" :
Rubberglue : Product)
```

- What the DBMS does

What the DBMS does

- CREATE TYPE CUSTOMER AS ("Name" char, ADDRESS char) NodeType
- INSERT INTO CUSTOMER VALUES('JOE','Joe Edwards','10 Station Rd.')
- CREATE TYPE "Order" NodeType
- INSERT INTO "Order" VALUES('ORD201')
- CREATE TYPE ORDERED as ("Date" date) EdgeType (CUSTOMER, "Order")
- INSERT INTO ORDERED VALUES ('554','JOE','ORD201',date'2002-11-22')
- CREATE TYPE PRODUCT NodeType
- CREATE TYPE WOODSCREW UNDER PRODUCT
- INSERT INTO WOODSCREW VALUES ('16/8x100’)
- CREATE TYPE ITEM as (QTY int) EdgeType("Order",PRODUCT)
- INSERT INTO ITEM VALUES('1004','ORD201','16/8x100',5)
- And so on. Also, we need index constraints (not illustrated)

- Using SQL for definition

IARIA

Using SQL to define graphs

- Node and edge types can be created and modified using CREATE TYPE and ALTER TYPE by adding the metadata NODETYPE or EDGETYPE(leaving,arriving)
- If N is a node type, INSERT into N works, as does UPDATE and DELETE, and similarly for edge types
- SELECT from node and edge types works
- A good strategy is to predefine data types using SQL and then use CREATE to build the graph

Using MATCH

- The Neo4j MATCH statement is available

```
MATCH graph [where] [statement]
```

- The graph part is as in CREATE, except that dummy identifiers can be used for nodes and edges, preceded by _
- The result of MATCH is a table of possible values for these identifiers such that the graph fragment is found in the database
- Subject to the where condition if any
- The optional statement says what is to be done with these values, otherwise they are returned like in SELECT
- We can also use MATCH as a source of data for SELECT and INSERT

- An example

IARIA

A MATCH example

```
SQL> match (_)-[:Item {Qty:_Q}]->(_Y:_T) where Q>4
```

```
SQL> match (_)-[:Item {Qty:_Q}]->(_Y:_T) where Q>4
|-|-----------|----------|
|Q|Y          |T         |
|-|-----------|----------|
|5|16/8x100   |WOODSCREW|
|5|Fibre 12cm|WALLPLUG |
|-|-----------|----------|
```

Integrating MATCH and SQL

- Match can be used as a query (as in the last slide)
- Match can be used as a subquery for predicates etc (not yet for joins)
- Match can supply rows to be inserted in another table

Insert into T (MATCH ..)

- Extra work done by the DBMS

Extra work done by the DBMS

- Node and Edge ids need to be unique so the DBMS has an index for this
- The DBMS also keeps a list of the connected graphs to speed up searching
- MATCH statements address the entire database

Predefining Graph types

- create type student as (matric char) nodetype
- insert into student values ('Fred','22/456')

```
SQL> match(_S:Student)
|----|
|S   |
|----|
|Fred|
|----|
SQL> match(_S:Student{Matric:_M})
|----|------|
|S   |M     |
|----|------|
|Fred|22/456|
|----|------|
```

Transforming types

- create type person nodetype
- alter type student set under person

```
SQL> select * from person
|----|
|ID  |
|----|
|Fred|
|----|
SQL> select *,specifictype() from person
|----|------------|
|ID  |SPECIFICTYPE|
|----|------------|
|Fred|STUDENT     |
|----|------------|
SQL>
```

Extending node types

- create type staff under person as (title char)
- insert into staff values ('Anne','Prof')
- select *,specifictype() from person

```
SQL> select *,specifictype() from person
|----|------------|
|ID  |SPECIFICTYPE|
|----|------------|
|Anne|STAFF       |
|Fred|STUDENT     |
|----|------------|
SQL> select * from staff
|----|-----|
|ID  |TITLE|
|----|-----|
|Anne|Prof |
|----|-----|
SQL>
```

Friends of Friends

```
create type friend
edgetype(person,person)
[create trigger sym after insert on
friend referencing new as nr for each
row
if not exists (select id from friend
where leaving=nr.arriving and
arriving=nr.leaving)
then insert into
friend(leaving,arriving) values
(nr.arriving,nr.leaving) end if]
```


IARIA

Symmetric edges

```
insert into person
values('Joe'),('Mary’)
```

```
insert into friend(leaving,arriving)
values('Joe','Mary'),('Mary','Fred')
```

```
select id from friend where
leaving='Fred'
```

```
SQL> select id from friend where leaving='Fred'
|-----|
|ID   |
|-----|
|2426|
|-----|
```

Conclusions

- This merging of TGM with relational technology allows graph oriented data manipulation and queries
- Some realistic examples of the approach would be nice
- Extra graph-oriented syntax may be helpful, and metadata for multiplicity
- There is a potential for supporting interactive data modeling

- References

IARIA

References

1. F. Laux and M. Crowe, Information Integration using the Typed Graph Model. DBKDA 2021: The Thirteenth International Conference on Advances in Databases, Knowledge, and Data Applications, IARIA, May 2021, pp 7-14, ISSN 2308-4332, ISBN 978-1-61208-857-0
2. F. Laux, "The Typed Graph Model", DBKDA 2020 : The Twelfth International Conference on Advances in Databases, Knowledge, and Data Applications, IARIA, Sept 2020, pp. 13-19, ISSN: 2308-4332, ISBN: 978-1-61208-790-0
3. M. Crowe, and F. Laux, "Database Technology Evolution", IARIA International Journal on Advanced is Software, vol 15 (3-4) 2022, pp. 224-234, ISSN: 1942-2628

IARIA