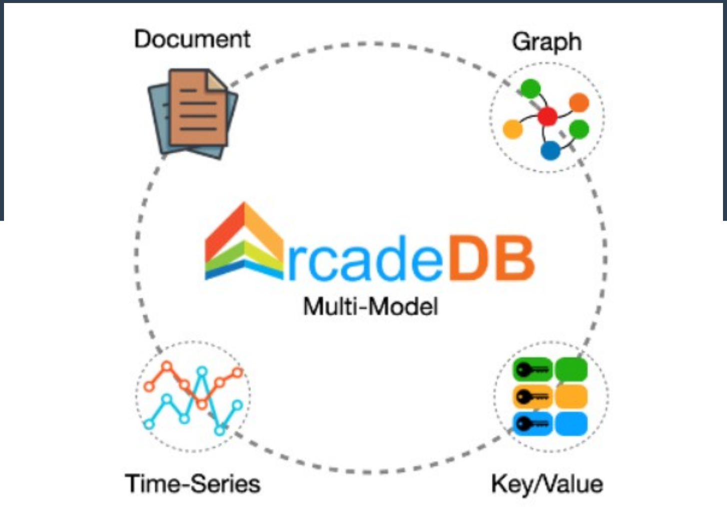
ArcadeDB

Jaroslav Lukniš

Introduction

- New generation of DBMS
- Released in 2021
- Written in Low-Level-Java (Java8+), without using high-level API
- Able to run on every sw/hw configuration
- Can run as embedded with language that runs Java Virtual Machine or can run with Docker, Kubernetes or just by running server script
- Free open source

- **Multi-Model: engine supports Graph, Document, Key/Value and Time-Series models**
- **Fast and scalable**
- **Originally forked from OrientDB**
- **Supports schema-less, schema-full and mixed modes**
- **Supports multiple languages: SQL, Cypher, Gremlin, GraphQL, MongoDB**



Example to execute a query by using GraphQL:

```graphql
{graphql}{ bookById(id: "book-1"){ id name authors { firstName, lastName } }
```

Example to use Cypher:

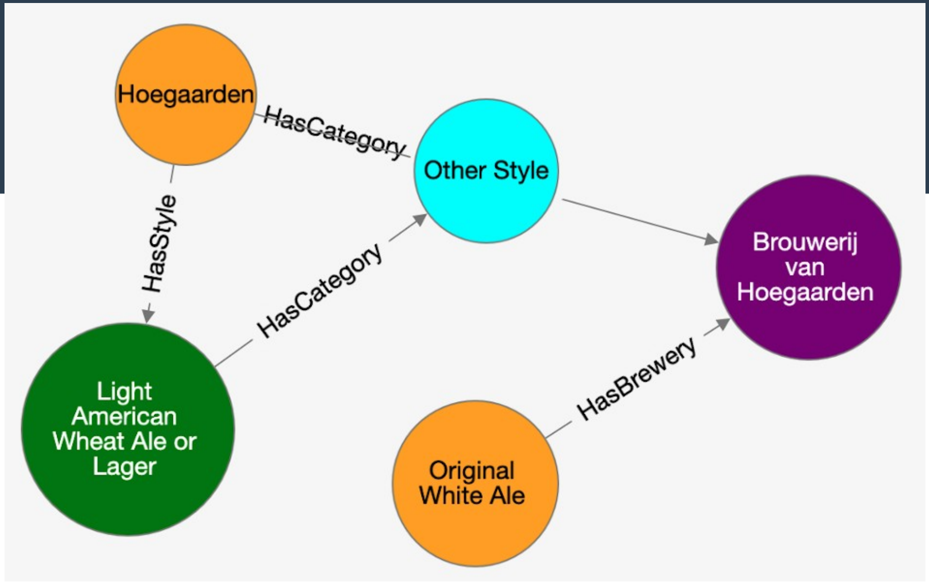
```
{cypher}MATCH (m:Movie)<-[a:ACTED_IN]-(p:Person) WHERE id(m) = '#1:0' RETURN *
```

Example of using Gremlin:

```
{gremlin}g.V()
```

Graph Model

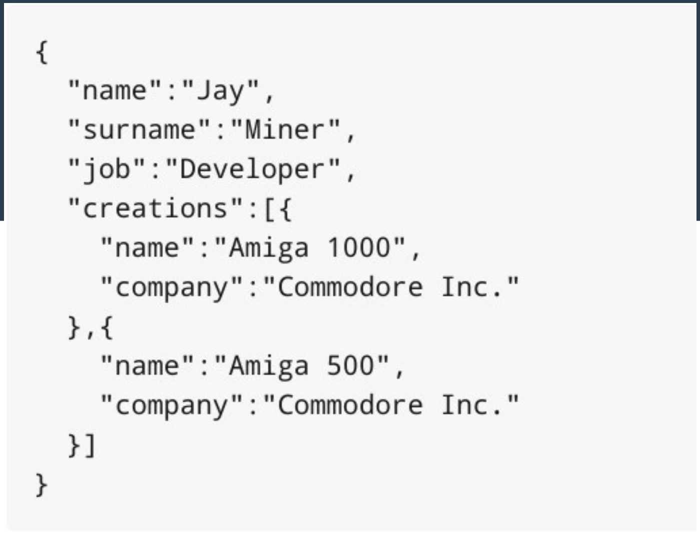
- **Model is represented by concept of property graph, which defines:**
- **Vertex – an entity, linked with other Vertices, mandatory properties:**
  - Unique identifier
  - Set of incoming/outgoing Edges
- **Edges - an entity, links two Vertices, mandatory properties:**
  - Unique identifier
  - Link to incoming Vertex (head)
  - Link to outgoing
  - Label defining type of connection/relationship
- **Also they can have custom properties defined by user**



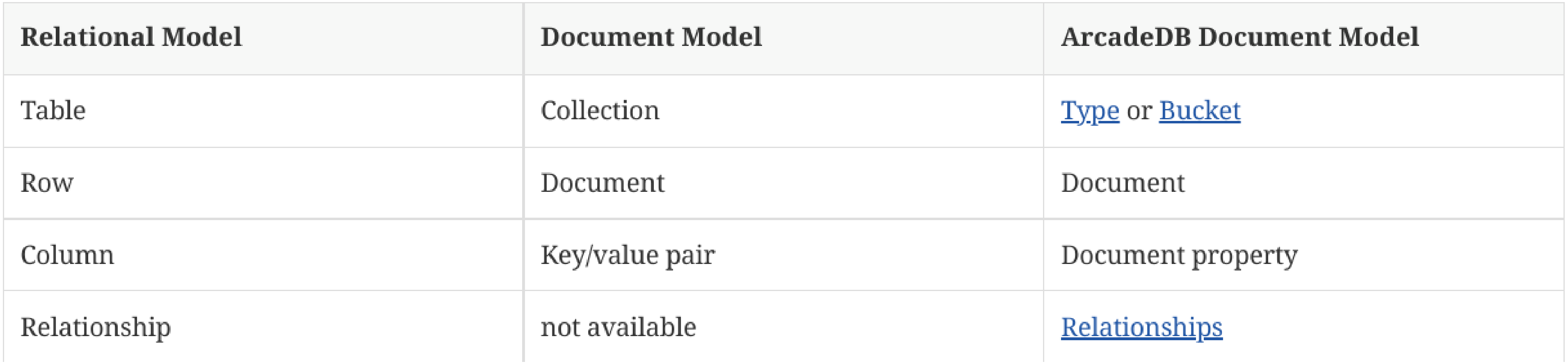
Relational Model	Graph Model	ArcadeDB Graph Model
Table	Vertex and Edge Types	Type
Row	Vertex	Vertex
Column	Vertex and Edge property	Vertex and Edge property
Relationship	Edge	Edge

Document Model

- document is a set of key/value pairs
- not typically forced to have a schema
- flexible and easy to modify
- documents are stored in collections, ArcadeDB uses Types and Buckets as form of collections
- adds the concept of a "Link" as a relationship between documents
- With ArcadeDB, you can decide whether to embed documents or link to them directly, when you fetch a document, all the links are automatically resolved by ArcadeDB



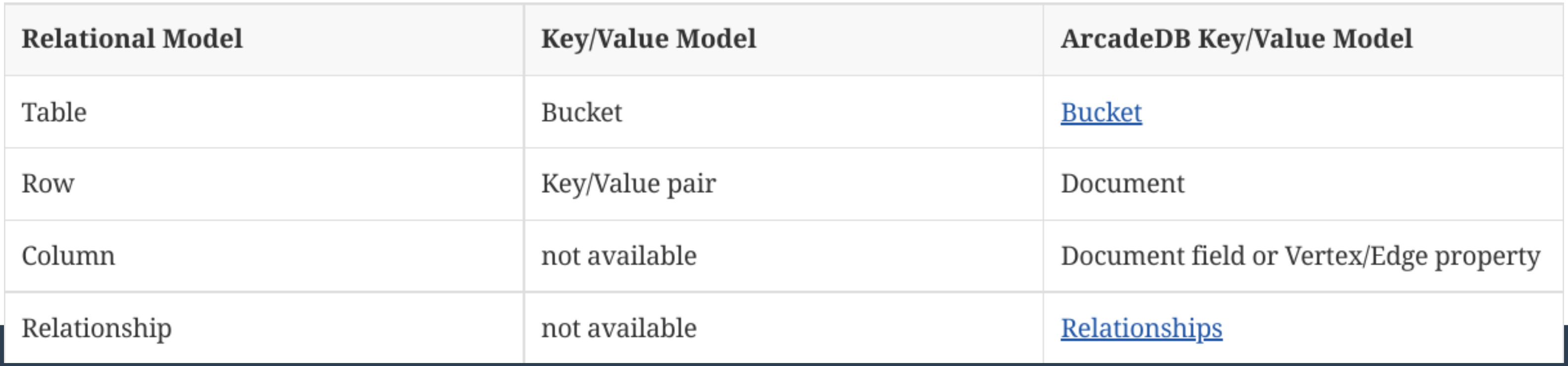
```
{
  "name":"Jay",
  "surname":"Miner",
  "job":"Developer",
  "creations":[{
    "name":"Amiga 1000",
    "company":"Commodore Inc."
  },{
    "name":"Amiga 500",
    "company":"Commodore Inc."
  }]
}
```



Relational Model	Document Model	ArcadeDB Document Model
Table	Collection	Type or Bucket
Row	Document	Document
Column	Key/value pair	Document property
Relationship	not available	Relationships

Key/Value Model

- Everything can be reached by key, values can be simple and complex types
- ArcadeDB allows graph elements and documents as values for richer model
- model provides "buckets" to group key/value pairs in different containers



Relational Model	Key/Value Model	ArcadeDB Key/Value Model
Table	Bucket	Bucket
Row	Key/Value pair	Document
Column	not available	Document field or Vertex/Edge property
Relationship	not available	Relationships

Main Concepts

- **Record – smallest unit, come in 3 types: Document, Vertex, Edge**
- **Document – schema-full/schema-less, handle fields in flexible manner, import/export in JSON format**
- **Vertex (Nodes) – main entity with information, additional features/properties, connected with Edges**
- **Edges (Arcs) – connection between Vertices, unidirectional/bidirectional**
- **RecordID (RID) – auto-generated unique identifier for record, immutable, universal, never reused, access by RID in O(1)complexity, format #<bucket-identifier>:<record-position>,**
  - Bucket-identifier – id of bucket to which record belongs (max 2,147,483,643 buckets in database)
  - Record-position – absolute position of record in bucket (#-1:-1 is null RID)

- **Types – closest to ‘Table’ in relational databases, schema-less/schema-full/mix, can inherit attributes from other types**
  - Each type has buckets (data files), one type can have multiple buckets, query against type fetches all buckets
- **Buckets - provide physical or in-memory space in which ArcadeDB actually stores the data, bucket = one file at file system, part of one type, significant help during queries**
- **Relationships -**
  - Referenced – storing direct links to target object
  - Embedded – storing the relationship within the record, stronger than referenced
    - 1:1/n:1 – express using EMBEDDED type
    - 1:n/n:n – express using LIST (ordered list) or MAP (ordered map key:value) type
- **Transactions – ACID: atomicity (all or nothing), consistency (from one valid state to another), isolation (incomplete transaction might not even be visible to another), durability (committed transactions will remain)**

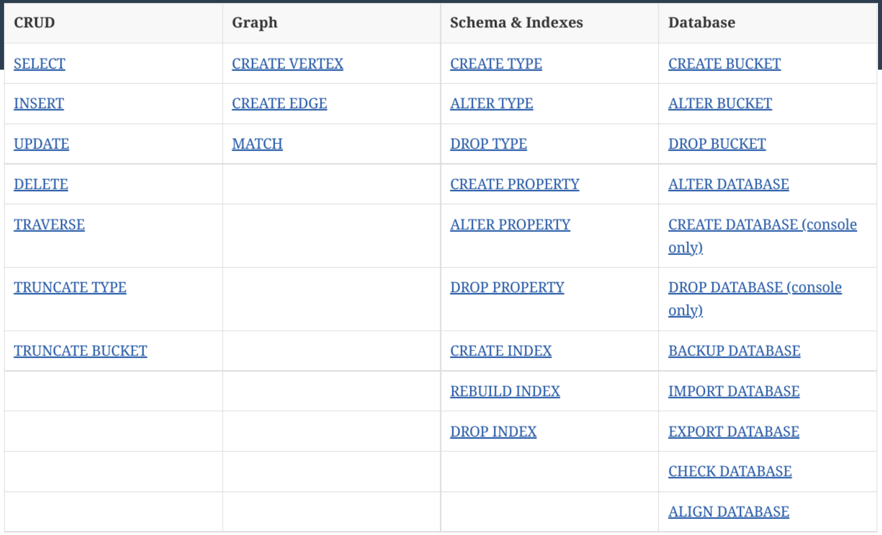
```
CREATE BUCKET Customer_Europe
CREATE BUCKET Customer_Americas
CREATE BUCKET Customer_Asia
CREATE BUCKET Customer_Other

CREATE VERTEX TYPE Customer BUCKET Customer_Europe,Customer_Americas,Customer_Asia,Customer_Other
```

```
Customer Record A -------------> Record B Invoice
         RID #5:23                  RID #10:2
```

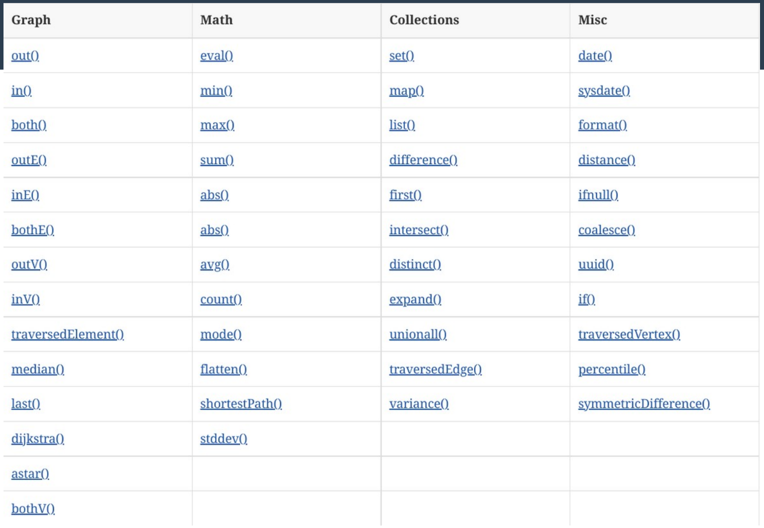
```
     Record A <>----------> Record B
    TYPE=Account          TYPE=Address
     RID #5:23               NO RID
```

```
arcadeDB> SELECT FROM Account WHERE address.city = 'Rome'
```

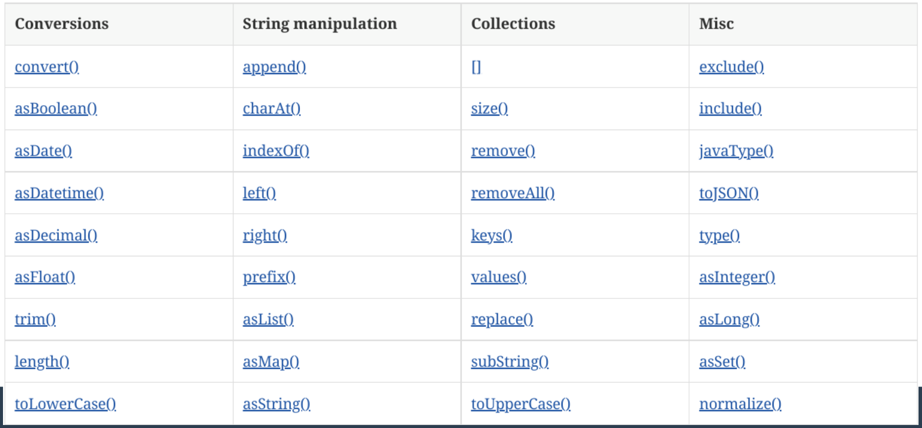
Commands, functions, methods



CRUD	Graph	Schema & Indexes	Database
SELECT	CREATE VERTEX	CREATE TYPE	CREATE BUCKET
INSERT	CREATE EDGE	ALTER TYPE	ALTER BUCKET
UPDATE	MATCH	DROP TYPE	DROP BUCKET
DELETE		CREATE PROPERTY	ALTER DATABASE
TRAVERSE		ALTER PROPERTY	CREATE DATABASE (console only)
TRUNCATE TYPE		DROP PROPERTY	DROP DATABASE (console only)
TRUNCATE BUCKET		CREATE INDEX	BACKUP DATABASE
		REBUILD INDEX	IMPORT DATABASE
		DROP INDEX	EXPORT DATABASE
			CHECK DATABASE
			ALIGN DATABASE



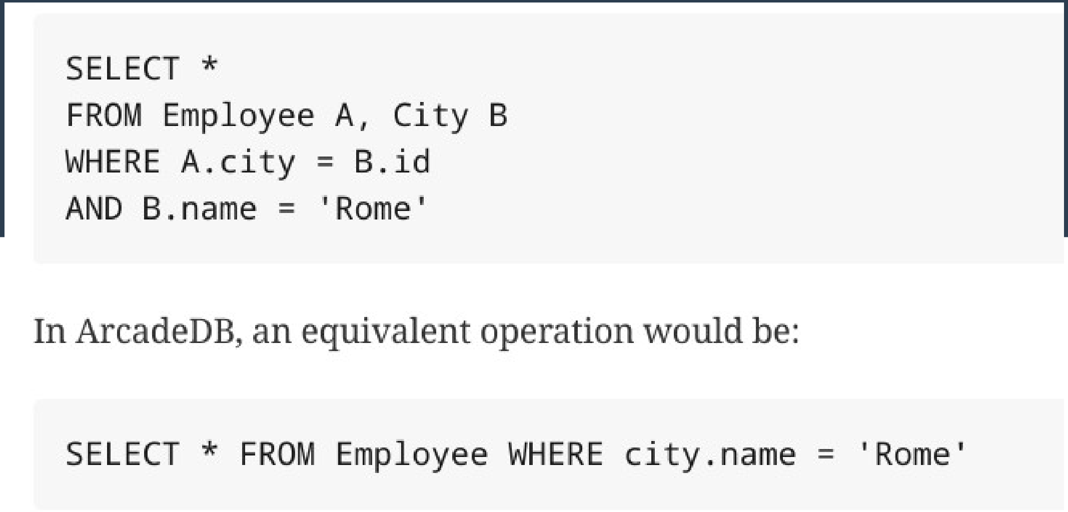
Graph	Math	Collections	Misc
out()	eval()	set()	date()
in()	min()	map()	sysdate()
both()	max()	list()	format()
outE()	sum()	difference()	distance()
inE()	abs()	first()	ifnull()
bothE()	abs()	intersect()	coalesce()
outV()	avg()	distinct()	uuid()
inV()	count()	expand()	if()
traversedElement()	mode()	unionall()	traversedVertex()
median()	flatten()	traversedEdge()	percentile()
last()	shortestPath()	variance()	symmetricDifference()
dijkstra()	stddev()		
astar()			
bothV()			



Conversions	String manipulation	Collections	Misc
convert()	append()	[]	exclude()
asBoolean()	charAt()	size()	include()
asDate()	indexOf()	remove()	javaType()
asDatetime()	left()	removeAll()	toJSON()
asDecimal()	right()	keys()	type()
asFloat()	prefix()	values()	asInteger()
trim()	asList()	replace()	asLong()
length()	asMap()	subString()	asSet()
toLowerCase()	asString()	toUpperCase()	normalize()

Commands, functions, methods

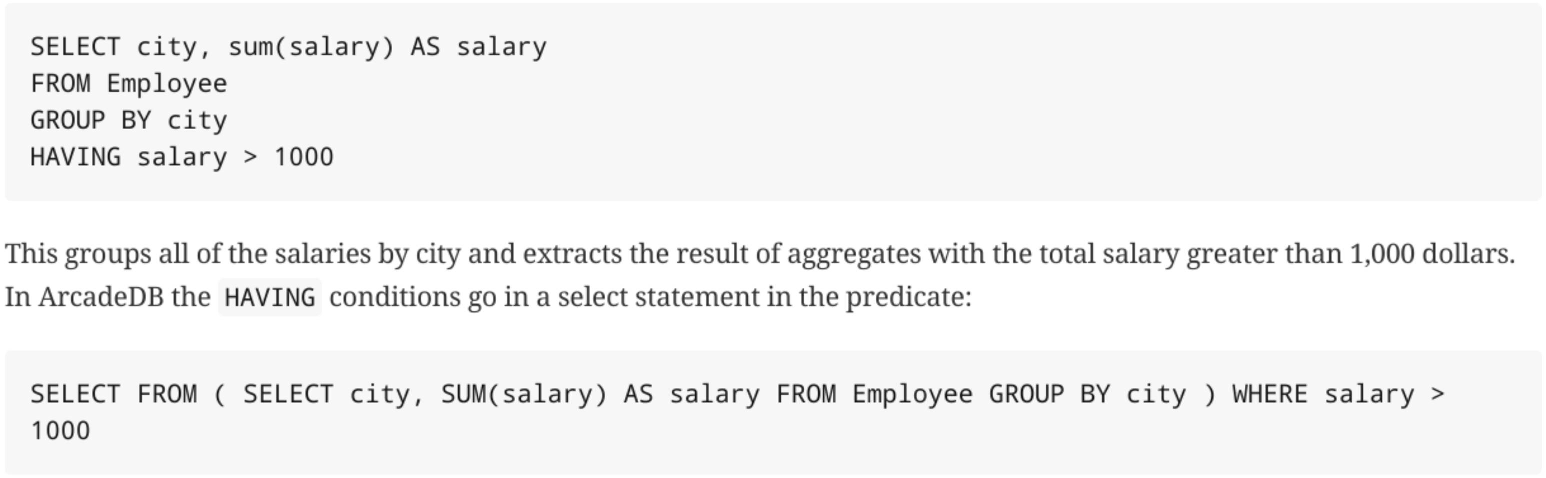
- No JOINS – relationships represented by LINKS
- No “HAVING” keyword, instead nested queries
- Supports selection, projection, aliases, conditions, operators, where clause, grouping, ordering, pagination, matching, batch (executing more commands), transactions
- Commands EXPLAIN, UPDATE, PROFILE (similar to EXPLAIN)



```
SELECT *
FROM Employee A, City B
WHERE A.city = B.id
AND B.name = 'Rome'
```

In ArcadeDB, an equivalent operation would be:

```
SELECT * FROM Employee WHERE city.name = 'Rome'
```



```
SELECT city, sum(salary) AS salary
FROM Employee
GROUP BY city
HAVING salary > 1000
```

This groups all of the salaries by city and extracts the result of aggregates with the total salary greater than 1,000 dollars. In ArcadeDB the `HAVING` conditions go in a select statement in the predicate:

```
SELECT FROM ( SELECT city, SUM(salary) AS salary FROM Employee GROUP BY city ) WHERE salary >
1000
```

- Create an edge of the type `E1` and define its properties:

```
ArcadeDB> CREATE EDGE E1 FROM #10:3 TO #11:4 SET brand = 'fiat', name = 'wow'
```

- Create edges of the type `Watched` between all action movies in the database and the user Luca, using sub-queries:

```
ArcadeDB> CREATE EDGE Watched FROM (SELECT FROM account WHERE name = 'Luca') TO
            (SELECT FROM movies WHERE type.name = 'action')
```

```
ArcadeDB> DELETE FROM Profile WHERE surname.toLowerCase() = 'unknown'
```

```
ArcadeDB> INSERT INTO Profile SET name = 'Jay', surname = 'Miner'
```

```
ArcadeDB> INSERT INTO Profile CONTENT {"name": "Jay", "surname": "Miner"}
```

```
ArcadeDB> MATCH {type: Person, as: person, where: (name = 'John')}.both('Friend') {as: friend}
            RETURN person, friend

--------+---------
 person | friend
--------+---------
 #12:0  | #12:1
 #12:0  | #12:2
 #12:0  | #12:3
 #12:1  | #12:0
 #12:1  | #12:2
--------+---------
```

- Update an embedded document. The `UPDATE` command can take JSON as a value to update.

```
ArcadeDB> UPDATE Account SET address={ "street": "Melrose Avenue", "city": {
               "name": "Beverly Hills" } }
```

```
ArcadeDB {db=foo}> explain select from v where name = 'a'

Profiled command '[{

executionPlan:{...},

executionPlanAsString:

+ FETCH FROM TYPE v
  + FETCH FROM BUCKET 9 ASC
  + FETCH FROM BUCKET 10 ASC
  + FETCH FROM BUCKET 11 ASC
  + FETCH FROM BUCKET 12 ASC
  + FETCH FROM BUCKET 13 ASC
  + FETCH FROM BUCKET 14 ASC
  + FETCH FROM BUCKET 15 ASC
  + FETCH FROM BUCKET 16 ASC
  + FETCH NEW RECORDS FROM CURRENT TRANSACTION SCOPE (if any)
+ FILTER ITEMS WHERE
  name = 'a'

}]' in 0,022000 sec(s):
```

```
PROFILE SELECT sum(Amount), OrderDate
FROM Orders
WHERE OrderDate > date("2012-12-09", "yyyy-MM-dd")
GROUP BY OrderDate
```

result:

```
+ FETCH FROM INDEX Orders.OrderDate (1.445μs)
  OrderDate > date("2012-12-09", "yyyy-MM-dd")
+ EXTRACT VALUE FROM INDEX ENTRY
+ FILTER ITEMS BY TYPE
  Orders
+ CALCULATE PROJECTIONS (5.065μs)
  Amount AS _$$$OALIAS$$_1, OrderDate
+ CALCULATE AGGREGATE PROJECTIONS (3.182μs)
  sum(_$$$OALIAS$$_1) AS _$$$OALIAS$$_0, OrderDate
  GROUP BY OrderDate
+ CALCULATE PROJECTIONS (1.116μs)
  _$$$OALIAS$$_0 AS `sum(Amount)`, OrderDate
```

Pros/Cons

- **Pros**
  - Fast, flexible
  - Automatic usage of indexes
  - Possible usage of Studio (web tool)
  - Usage from command line
  - Interaction from multiple APIs
  - Multi-model
  - Backup/restore databases
- **Cons**
  - New, so less informations
  - Little complicated setup
  - Still new, so some issues can appear

Summary

- Properties
- Informations about models
- Languages, queries, main concepts

Thank you for your attention