Distributed Immutable Data Object Command Line Interface DIDO-CLI

R. W. “Nick” Stavros
Bryan A. Turek
Ian T. Stavros
Jackrabbit Consulting, Inc.
30 January 2020

What is DIDO (Distributed Immutable Data Object)?

- It represents a class of concepts, activities and products that have become the trend
- Each word within the name is important
  - ***Distributed*** – the class is distributed across many computers
  - ***Immutable*** – the data it represents once written cannot be changed (write once, read many times)
  - ***Data*** – at the core of the class is always the Data
  - ***Object*** – the data are encapsulated as objects with a uniform set of operations

Why not Blockchain or Distributed Ledger Technologies?

- These terms refer to the technology used to accomplish a DIDO
- All ***Blockchains*** or ***Distributed Ledgers Technologies (DLT)*** are DIDOs but the inverse is not true
- Newer DIDO implementations such as Iota's ***Directed Acyclic Graphs (DAGs)*** may not fit the "traditional" definition of blockchains or DLTs.
- As an analogy, using words like Blockchain or DLT are like using B-Tree or Hashmaps to describe a datastore. ***These are technologies*** used by datastores.

What are DIDO Platforms

- ***Software Platform*** *provides low-level functionality ready-made as an accelerator to a consumable solution.* https://bridgera.com/what-is-a-software-platform/
- ***DIDO Platform*** *is a software platform providing the low-level functionality of ready-made to accelerate the adoption of DIDOs*

What is a Command Line Interface (CLI)

- ***Command Line Interface (CLI)*** **is a text-based interface that is used to operate software and operating systems while allowing the user to respond to visual prompts by typing single commands into the interface and receiving a reply in the same way.**
- **CLI is quite different from the *Graphical User Interface (GUI*) that is presently being used in the latest operating systems.**
  https://www.techopedia.com/definition/3337/command-line-interface-cli

What is an Application Programming Interface (API)?

An Application Programming Interface (API) is a set of protocols, routines, functions and/or commands that programmers use to develop software or facilitate interaction between distinct systems. APIs are available for both desktop and mobile use, and are typically useful for programming GUI (graphic user interface) components, as well as allowing a software program to request and accommodate services from another program.

What is Data Definition Language (DDL)

A *Data Definition Language (DDL)* is a computer language used to create and modify the structure of database objects in a database. These database objects include views, schemas, tables, indexes, etc.

This term is also known as *Data Description Language* in some contexts, as it describes the fields and records in a database table.

What is Data Manipulation Language (DML)

*A **Data Manipulation Language (DML)** is a family of computer languages including commands permitting users to manipulate data in a database. This manipulation involves inserting data into database tables, retrieving existing data, deleting data from existing tables and modifying existing data. DML is mostly incorporated in SQL databases.*

Setting the Stage (1 of 2)

- **Why are DBMSs, DDL and DML important in the context of DIDOs?**

  Because the evolutionary path these current datastore products is analogous to the evolution of DIDOs and should be considered a roadmap of where to go

- **What is the difference between a DBMS and a Datastore?**

  A datastore is a repository for persistently storing and managing collections of data which include not just repositories like databases, but also simpler store types such as simple files, emails etc.

Setting the Stage (2 of 2)

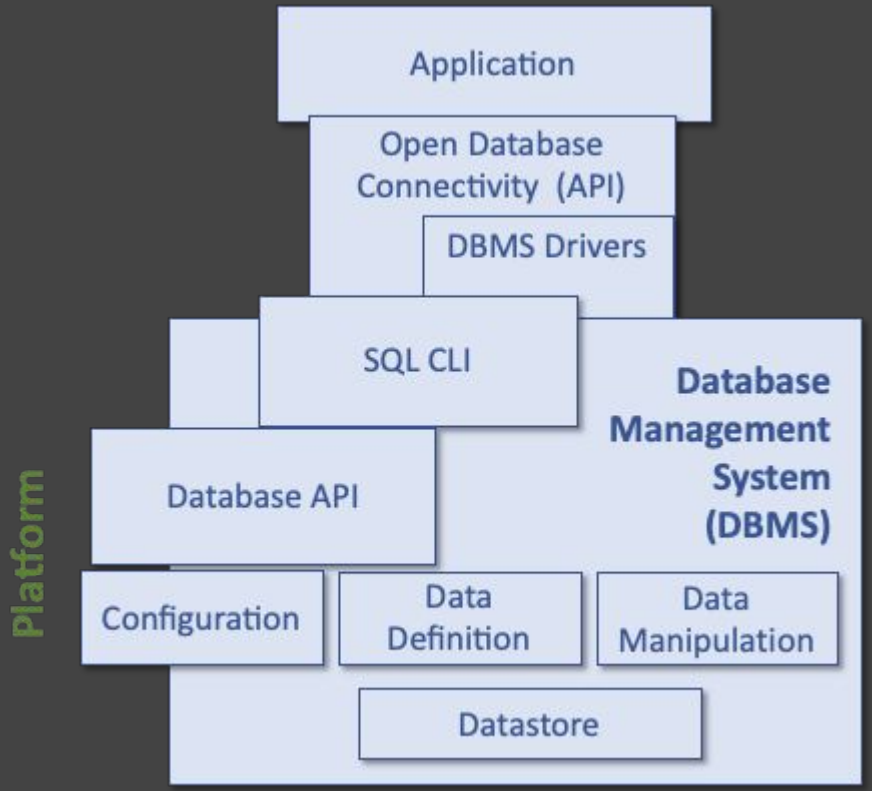
- **What is the difference between a DBMS and a Datastore?**

  A datastore is a repository for persistently storing and managing collections of data which include not just repositories like databases, but also simpler store types such as simple files, emails etc. http://www.information-management.com/glossary/d.html

- **Does this mean DIDOs support Update operations?**

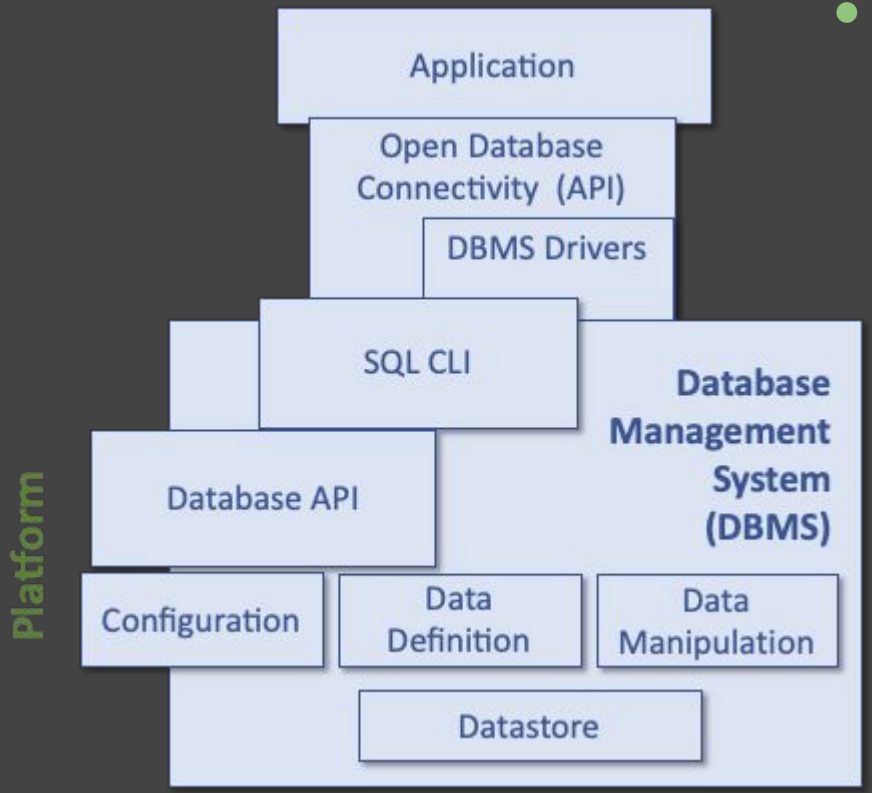
  Yes and No! They support updates through a transaction based "Ledger" … they are Immutable!

Typical Mature Database “stack” (1 of 5)



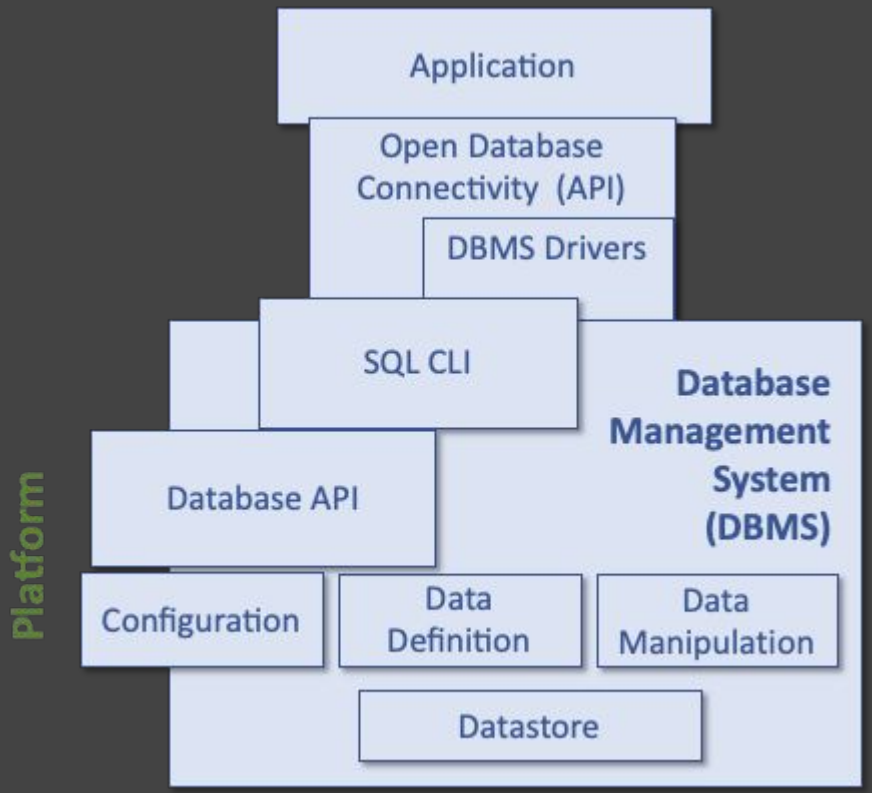
- *The term “stack” is in quotes because there is not a single path. The blocks are more like components and they are “glued” together depending on the context*
- *At the heart of the DBMS is the Datastore, not dissimilar to DIDOs*

Typical Mature Database “stack” (2 of 5)



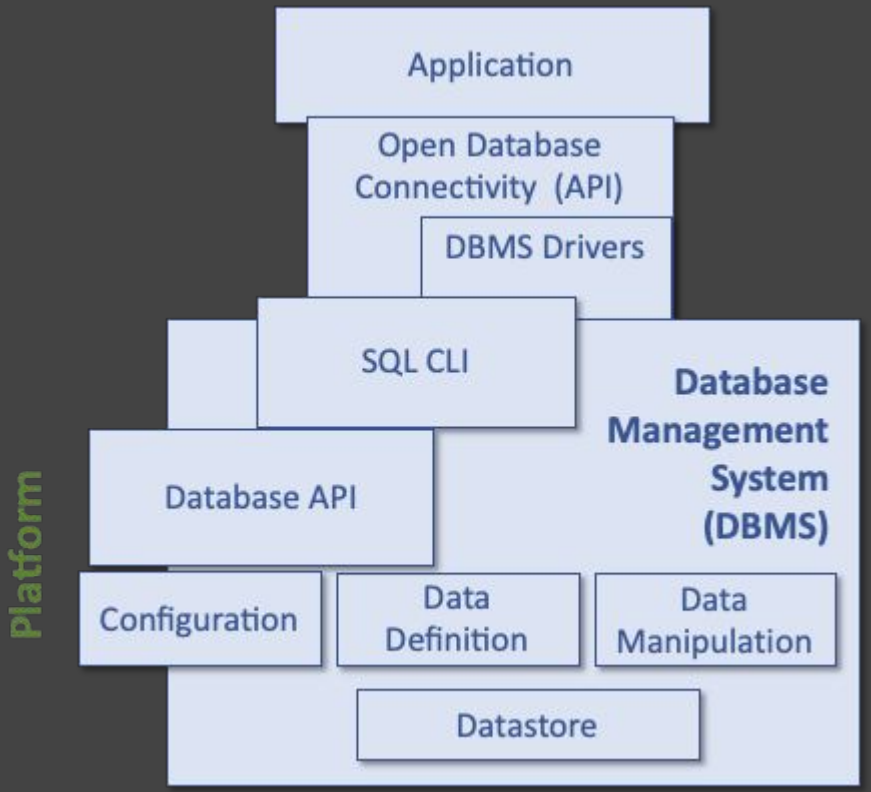
- *Applications are generally built upon Open Database Connectivity “Layer”*
  - *With specific driver to access a specific database*
    - *Using standardized SQL API*
    - *DB Specific API*

Typical Mature Database “stack” (3 of 5)



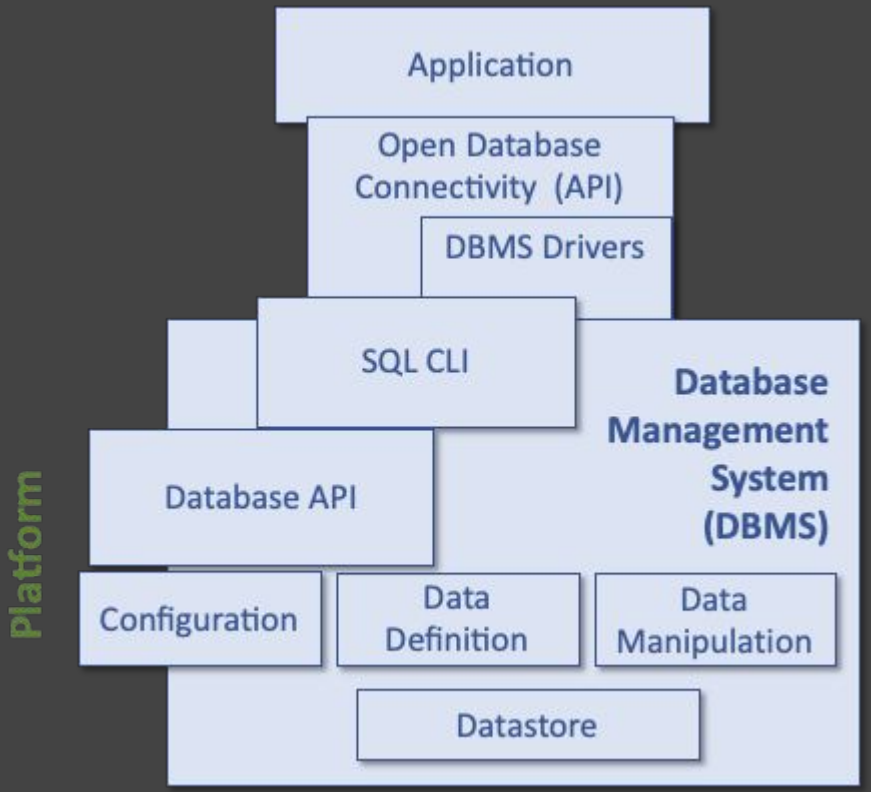
- *A typical DBMS provides many APIs*
  - *Database Configuration*
  - *Data Definition*
  - *Data Manipulation*
  - *Custom DBMS API*
  - *SQL Command Line Interpreter*
- *Which one are we going to build or standardize first?*

Typical Mature Database “stack” (4 of 5)



- *Most modern Applications formulate SQL statements as strings to access the DBMS*
- *The SQL strings are passed through an Open Data Connectivity API (i.e., ODBC)*

Typical Mature Database “stack” (5 of 5)



- *The ODBC either passes the SQL Strings to the DBMS SQL CLI or through DBMS Drivers written specifically for the DBMS*
- *DBMS Drivers either send commands to the SQL CLI Interpreter or they call DBMS specific APIs*
- *Results are SQLSTATES (i.e., return codes), Text, or Results Sets*

Using the DBMS Evolution as a Roadmap for DIDOs

- The components of the DBMS APIs should map nicely to those required by DIDOs
  - Configuration
  - Definition
  - Manipulation
- Providing a vendor-neutral Command Line Interface (CLI) will abstract the details of a DIDO imnplementation from the end users (who are the end users?)

DIDO Configuration

Issue:

DIDO's run on distributed computers owned by many people or organizations, each one has a varying amount of Information Technology (IT) support and a unique configuration

Problem:

There are no assurances as to the state of the machines (including security) that the DIDO run on or even how the DIDO is installed on the machine

Containerization and Virtual Machines

Solution:

Using Containers or Virtual Machines (VMs) to deploy multiple, isolated services on a single platform.

What's the difference?

- *The Container's system requires an underlying operating system providing basic services to the containerized applications using virtual-memory for isolation.*
- *A VM uses a hypervisor with its own operating system (OS) and using hardware virtualization*
- *Containers have lower overhead typically targeting environments with constrained resources*
- *Containers are usually isolated from other containers and have limited resource access. i.e., file systems or network support*

DIDO Configuration Definition Language (DCDL)

- *Containers and VMs have well defined, formal configuration definitions*
- *Create a DIDO CLI named Configuration Definition Language (DCDL)*
- *Considering some basic DIDO objects*
  - *Ports*
  - *Machine/Container resources*
  - *Disk Access*
  - *Service Images*
  - *Executable Images*
  - *DIDO configurations*
  - *Objects*
  - *Wallets*
  - *Exchanges*
  - *Oracles*
  - *Aggregates*
- *Considering some basic DIDO operations*
  - *Install / Download / Remove*
  - *Start / Stop / Pause / Resume*
  - *Synchronize*
  - *Dump Trace*
  - *Show*

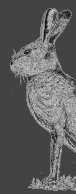
DCDL PORTS, RESOURCES, VOLUMES Examples (1 of 4)

```
DEFINE PORT http AS
     protocol  = TCP,
     port      = 8080;

DEFINE PORT discovery AS
     protocol  = UDP,
     port      = 30301;

DEFINE MACHINE_RESOURCE avaerageMachine AS
     Memory    = 64mi, MaxMemory = 100mi,
     CPU       = 100m, MaxCPU    = 110m;

DEFINE VOLUME config AS
       path      = "/etc/myDidoRoot/config";
DEFINE VOLUME data AS
     path      = "/etc/myDidoRoot/data";
DEFINE VOLUME download AS
     path      = "/etc/myDidoRoot/download";    -- Note: There is only one wallet VOLUME,
DEFINE VOLUME wallet AS                         -- It can point to different locations
     path      = "~/myWallet/";
DEFINE VOLUME wallet AS
     path      = "~/universityWallet/";
```

DCDL SERVICES, EXECUTABLES Examples (1 of 4)

```
DEFINE SERVICE PeerDiscovery AS
  image     = PeerDiscovery
  command   = ["/bin/sh", "-c"]
  arguments = "set -e -x; while true; do python "
              "/etc/peer_discovery.py; "
               "sleep 15; done";

DEFINE SERVICE fullNode AS
  image     = fullNode
  command   = ["/bin/sh", "-c"]
  arguments = "set -e -x; while true; do python "
              "/etc/full_node.py; "
              "sleep 15; done"

DEFINE EXECUTABLE todaysWeather AS
  image     = weather
  command   = ["/bin/sh", "-c"]
  arguments = "set -e -x; while true; do python "
              "/etc/weather.py; "
              "sleep 15; done"
```

DCDL DIDO Examples (2 of 4)

```
DEFINE DIDO MyDido
  USING
    port             = [ http, discovery ],
    machine_resource = avaerageMachine,
    volume           = [ config, data],
    service          = [ peerDiscovery, fullNode ],
    execute          = [ todaysWeather ];

DEFINE DIDO YourDido
  USING
    port             = [http,discovery],
    machine_resource = avaerageMachine,
    volume           = [ config, data],
    container        = [ peerDiscovery, fullNode ];

DEFINE DIDO OurDido
  USING
    Port             = [http,discovery],
    machine_resource = avaerageMachine,
    volume           = [ config, data],
    Container        = [ peerDiscovery, fullNode ];
```

```
DEFINE DIDO UniDido
  USING
    port             = [ http, discovery ],
    machine_resource = avaerageMachine,
    volume           = [ config, data],
    service          = [ peerDiscovery, fullNode ],
    execute          = [ getTodayWeather ];

DEFINE DIDO CourseListing
  USING
    port             = [ http, discovery ],
    machine_resource = avaerageMachine,
    volume           = [ config, data],
    service          = [ peerDiscovery, fullNode ],
    execute          = [ getTodayWeather ];

DEFINE DIDO Grades
  USING
    port             = [ http, discovery ],
    machine_resource = avaerageMachine,
    volume           = [ config, data],
    service          = [ peerDiscovery, fullNode ],
    execute          = [ getTodayWeather ];
```

DCDL DIDO Operation Examples (3 of 4)

```
DOWNLOAD DIDO MyDido
  FROM 'https://university.edu/studentDido/download/'
  ON  node                  = '2001:0db8:85a3:0000:0000:8a2e:0370:7334';
INSTALL DIDO MyDido
  ON  node                  = '2001:0db8:85a3:0000:0000:8a2e:0370:7334';
START DIDO MyDido
      ON node               = '2001:0db8:85a3:0000:0000:8a2e:0370:7334';
PAUSE DIDO MyDido
  ON  node                  =  '2001:0db8:85a3:0000:0000:8a2e:0370:7334';
DUMP TRACE DIDO MyDido
  ON  node                  =  '2001:0db8:85a3:0000:0000:8a2e:0370:7334';
RESUME DIDO MyDido
  ON  node                  =  '2001:0db8:85a3:0000:0000:8a2e:0370:7334';
REMOVE DIDO MyDido
  ON  node                  =  '2001:0db8:85a3:0000:0000:8a2e:0370:7334';

-- Note: the IP Address '2001:0db8:85a3:0000:0000:8a2e:0370:7334'
-- can be replaced by localhost
-- For example:
DOWNLOAD DIDO MyDido
  FROM 'https://university.edu/studentDido/download/'
  ON  node                  = localhost
```

DCDL DIDO Operation Examples (1 of 4)

```
START DIDO MyDido
      ON node                  = '2001:0db8:85a3:0000:0000:8a2e:0370:7334';
START DIDO YourDido
      ON node                  = '2001:0db8:85a3:0000:0000:8a2e:0370:7335';
START DIDO OurDido
      ON node                  = '2001:0db8:85a3:0000:0000:8a2e:0370:7336';
START DIDO UniDido
      ON node                  = '2001:0db8:85a3:0000:0000:8a2e:0370:7337';
START DIDO CourseListing
      ON node                  = '2001:0db8:85a3:0000:0000:8a2e:0370:7338';
START DIDO Grades
      ON node                  = '2001:0db8:85a3:0000:0000:8a2e:0370:7339”;

-- Note: Using containers, all these could be started on the same node.
```

DCDL SHOW Operation Examples (1 of 4)

```
SHOW PORT *;
SHOW PORT http;
SHOW SERVICE *;
SHOW SERVICE fullNode;
SHOW EXECUTE getTodayWeather;
SHOW DIDO *;
SHOW DIDO MyDido;
```

DIDO Data Definition Language (DDDL) (1 of 6)

- *Create a DIDO CLI named Data Definition Language (DDDL)*
- *Considering some basic DIDO objects*
  - *Objects*
  - *Wallets*
  - *Exchanges*
  - *Oracles*
    - *Between DIDOs*
    - *External*
  - *Aggregates*
- *Considering some basic DIDO operations*
  - *Create*
  - *Alter - a form of Aggregate between the original object and the new object*
  - *Delete - it is not possible to delete DIDO objects, they can only be marked as deleted or deprecated*

DDDL Object Examples (2 of 6)

```
CREATE OBJECT MyDido.Student AS
  ( StudentId    : Hash  NOT NULL KEY,
    FirstName    : Text  NOT NULL,
    LastName     : Text  NOT NULL
  );
CREATE OBJECT OurDido.StudentAccount AS
  ( AccountId    : Hash  NOT NULL KEY,
    StudentId    : Hash  NOT NULL,
    Balance      : Fixed(3.2), DEFAULT 0.0, MINIMUM 0.00,
                   MAXIMUM 10000.00, TOKEN
  );
CREATE OBJECT OurDido.DinningHallAccount AS
  ( AccountId    : Hash  NOT NULL KEY,
    StudentId    : Hash  NOT NULL,
    Balance      : Fixed(3.2), DEFAULT 0.0, MINIMUM 0.00,
                   MAXIMUM 500.00, TOKEN
  );
```

```
CREATE OBJECT CourseListing.Course AS
  ( CourseId      : Hash  NOT NULL KEY,
    Department    : Text  NOT NULL,
    Title         : Text  NOT NULL,
    CourseNumber  : Text  NOT NULL,
    Credits       : Integer
  );

CREATE OBJECT CourseListing.Post AS
 ( PostId       : Hash NOT NULL KEY,
   CourseId     : Hash  NOT NULL,
   StudentId    : Hash  NOT NULL,
   Grade        : Enum,
          DEFAULT {0,'Audit'},
          VALUES {{0,'Audit'},
                  {1,'Fail'},
                  {2,'Unsatisfactory'},
                  {3,'Fair'}
                  {4,'Good'}
                  {5,'Excellent'},
                 }
 );
```

DDDL Exchange Examples (3 of 6)

```
CREATE EXCHANGE UniDido.TopOffDining
  ( FromAccount      Hash,
    ToAccount        Hash,
    AmountToTransfer Fixed
  ) AS
  FROM OurDido.StudentAccount
    TO OurDido.DinningHallAccount
       ( AccountsID   : Hash;
         StudentId    : FETCH genesis KEY
                          FROM OurDido.StudentAccount STUDENT_ACCT
                         WHERE STUDENT_ACCT.AccountId
                               = $FromAccount.;
         AccountId    : SELECT genesis KEY
                          FROM OurDido.DinningHallAccount DINING_ACCT
                          WHERE DINING_ACCT.AccountId
                                = $ToAccount.;
      )
      AS TRANSACTION -- Start Transaction
        ( TRANSFER FROM StudentId::current KEY
                     TO OurDido.DinningHallAccount
                 AMOUNT $AmountToTransfer.
                TIMEOUT = 10*60000 --- 10 minutes
```

```
WHEN SUCCESS
  THEN POST INFORMATION
    TO Log, Notify
 OBJECTS OurDido.StudentAccount,
         OurDido.DinningHallAccount
   TEXT = 'The amount of $AmountToTransfer. '
          'Successful '
          'between $FromDido. and $toDido.'
WHEN FAILURE
  THEN POST WARNING
    TO log, Notify, Alert,
  OBJECTS OurDido.StudentAccount,
          OurDido.DinningHallAccount,
          OurDido.Administrator
   TEXT = 'The amuount of $AmountToTransfer. '
          'failure '
          'between $FromDido. and $toDido.'
); -- End Transaction
```

DDDL DIDO to DIDO Oracle Example (4 of 6)

```
CREATE ORACLE OurDido.StudentStatement
  FROM OurDido.StudentAccount,
       MyDido.Student
        ( StudentStatementId : Hash KEY,
          StudentId          : Hash,
          AccountId          : Hash
        )
    AS
     ( NEW KEY,
       FETCH  current KEY
          FROM MyDido.Student
         WHERE MyDido.Student        = STUDENT.$1.,
       FETCH  current KEY
          FROM OurDido.StudentAccount
         WHERE OurDido.StudentAccount = ACCOUNT.$2.
     );
```

```
-- Note: The Student Statement uses positional
-- parameter notation (i.e., first $1. and second $2.)
-- rather than named notation (i.e., StudentId
--  and AccountId). To use named parameters,
-- add a parenthetical list after the CREATE
-- statement. For Example:
-- CREATE oracle OurDido.StudentStatement
--          ( StudentId hash, AccountId hash )

-- Note: In this example, the structure of the ORACLE
--  object is laid out first, then the values for those
-- columns are associated with the column names
-- using a position.
```

DDDL RESTful to DIDO Oracle Example (5 of 6)

```
CREATE ORACLE MyDido.StudentForecast
  FROM MyDido.todaysWeather,
       MyDido.Student
        ( StudentForecastId : Hash,
          StudentId : Hash
                    = FETCH current KEY
                        FROM MyDido.Student    STUDENT
                       WHERE STUDENT.StudentId = $1.,
          Forecast  : Hash
                    = SELECT FORMAT JSON ATTR *
                         FROM MyDido.todaysWeather WEATHER
                        WHERE WEATHER.Location = $2.
        );
```

```
-- NOTE: Forecast is a JSON string that contains
-- the forecast for the student at the specified
-- location.
-- MyDido.todaysWeather is an EXECUTABLE
-- defined previously
```

```
-- Note: In this example, the structure of the ORACLE
--  object is made and the values for those
-- columns are associated with the column names
-- at the same time..
```

DDDL AGGREGATE Example (6 of 6)

```
CREATE  AGGREGATE Grades.ReportCard
  FROM  CourseListing.Post   POSTING
 WHERE  POSTING.StudentId = '$1.'
RECORD POSTING current ATTR *             AS parents,
       Grades.ReportCard current ATTR *   AS process;
```

```
-- Note: In an AGGREGATE, in order to provide
-- pedigree, the parents of the new aggregate and
-- the process that generated the aggregate need
-- to be recoded.
```

DIDO Data Manipulation Language (DDML) (1 of 8)

- *Create a DIDO CLI named Data Manipulation Language (DDDL)*
- *Considering some basic DIDO objects*
  - *Objects*
  - *Wallets*
  - *Exchanges*
  - *Oracles*
  - *Aggregates*
- *Considering some basic DIDO operations*
  - *Select*
  - *Store*
  - *Invoke*
  - *Fetch*
  - *Refresh*
  - *Show*

DDML FETCH Example (2 of 8)

```
-- To get the current StudentStatment
-- call the 'oracle' with the
-- positional parameters
  FETCH current KEY
    FROM OurDido.StudentStatement '12FFCD34..'
                                  '67DEAD34..'
-- To get the current StudentStatment
-- call the 'oracle' with the named
-- positional parameters
      FETCH current KEY
       FROM OurDido.StudentStatement
            2='67DEAD34..'
            1='12FFCD34..'
```

```
-- To get the current StudentStatment call
-- the 'oracle' with the named
-- parameters
-- Change
--    from: ORACLE OurDido.StudentStatement
--      to: ORACLE OurDido.StudentStatement
              ( StudentId, AccountId );
-- Change
--    from: $1. to $StudentId.
--    from: $2. to $AccountId.
```

DDML FETCH examples (3 of 8)

```
-- GENESIS -  Returns the KEY of first record in the history.
--         Note: The first record could be an AGGREGATE which means it is composed
--               one or more parents. Each parent has a genesis record of its own
  FETCH genesis  KEY FROM MyDido.Student WHERE StudentId = '12FFCD34..';

-- PREVIOUS - Returns the KEY of the later record in time. If at the current key,
--            the current key is returned
  FETCH previous KEY FROM MyDido.Student WHERE StudentId = '12FFCD34..';

-- NEXT     - Returns the KEY of the earlier record in time. If at the genesis key,
--            the genesis key is returned
  FETCH next     KEY FROM MyDido.Student WHERE StudentId = '12FFCD34..';

-- CURRENT  - Returns the KEY of the most current record. If at the current key,
--            the current key is returned
  FETCH current  KEY FROM MyDido.Student WHERE StudentId = '12FFCD34..';

-- HISTORY  - Returns the set of KEYs in chronological order from the present location
--            key to the genesis. The history does not go back beyond the genesis entry
--            to include the parents if the object was an AGGREGATE
  FETCH history  KEY FROM MyDido.Student WHERE StudentId = '12FFCD34..';
```

DDML Initiate Example (4 of 8)

```
INITIATE MyDido.Student
                  ( FirstName = 'Robert',
                    LastName  = 'Stavros'
                  );
SINITIATE OurDido.StudentAccount
                  ( StudentId = '12FFCD34..',
                    Balance   =  500.00
                  );
INITIATE OurDido.DinningHallAccount
                  ( StudentId = '12FFCD34..',
                    Balance   =  0.00
                  );
```

```
-- Initiate creates a genesis record and
-- returns a key pair of a public and private
-- key.
--
-- This needs to be put into a Wallet or
-- into another DIDO object.
```

DDML Wallet Examples (5 of 8)

```
STORE INTO WALLET RobertsWallet
      Name      = 'MyDido.Student'
      KeyPair   = INITIATE MyDido.Student
                  ( FirstName = 'Robert',
                    LastName  = 'Stavros'
                  );
STORE INTO wallet:RobertsWallet
      Name      = 'OurDido.StudentAccount'
      KeyPair   = INITIATE OurDido.StudentAccount
                  ( StudentId = '12FFCD34..',
                    Balance   =  500.00
                  );
STORE INTO wallet:RobertsWallet
      Name      = 'OurDido.DinningHallAccount'
      KeyPair   = INITIATE OurDido.DinningHallAccount
                  ( StudentId = '12FFCD34..',
                    Balance   =  0.00
                  );
```

DDML INVOKE, REFRESH and STORE Examples (6 of 8)

```
INVOKE EXCHANGE UniDido.TopOffDining
  ( FromAccount      ='12FFCD34..',
    ToAccount        ='67DEAD34..',
    AmountToTransfer = 25
  );
REFRESH wallet:RobertsWallet;

STORE INTO universityWallet:CourseListingWallet
  Name     = 'CourseListing.Post',
  KeyPair  = INITIATE CourseListing.Course
               ( CourseId   = '66DDEEAADD..',
                 StudentId  = '12FFCD34..',
                 Grade      = 'A'
               );
STORE INTO universityWallet:CourseListingWallet
  Name     = 'CourseListing.Post'
  KeyPair  = INITIATE CourseListing.Course
              ( CourseId   = '77DFEEAADD..',
                StudentId  = '12FFCD34..',
                Grade      = 'C'
              );
```


DDML STORE Examples (7 of 8)

```
STORE INTO universityWallet:CourseListingWallet
      Name     = 'CourseListing.Post'
      KeyPair  = INITIATE CourseListing.Course
                 ( CourseId   = '66DDEEAADD..',
                   StudentId  = '12FFCD34..',
                   Grade      = 'A'
                 );

STORE INTO universityWallet:CourseListingWallet
      Name     = 'CourseListing.Post'
      KeyPair  = INITIATE CourseListing.Course
                 ( CourseId   = '77DFEEAADD..',
                   StudentId  = '12FFCD34..',
                   Grade      = 'C'
                 );
```

DDML STORE Examples (8 of 8)

```
STORE INTO universityWallet:CourseListingWallet
  Name      = 'CourseListing.Course'
  KeyPair   = Initiate CourseListing.Course
              ( Department   = 'Computer Science',
                Title        = 'Introduction',
                CourseNumber = 'CS-101'
                Credits      = 3
               );
      -- '54DFEEAADD..'
STORE INTO  universityWallet:CourseListingWallet
  Name      = 'CourseListing.Course'
  KeyPair   = INITIATE CourseListing.Course
              ( Department   = 'English',
                Title        = 'English Literature',
                CourseNumber = 'EN-101'
                Credits      = 3
                );
      -- '77DFEEAADD..'
```

```
STORE INTO universityWallet:CourseListingWallet
  Name      = 'CourseListing.Course'
  KeyPair   = Initiate CourseListing.Course
     ( Department = 'Computer Science',
       Title  = 'Text Information Systems',
       CourseNumber = 'CS-410'
       Credits      = 3
      );
   -- '66DDEEAADD..'
```

DIDO - CLI Conclusions

- It is not enough to provide a single APIs for DIDOs
  - Database Configuration
  - Data Definition
  - Data Manipulation
  - Custom DBMS API
  - Command Line Interpreter (CLI)
- Three different DIDO CLI are possible
  - DIDO Configuration Definition Language (DCDL)
  - DIDO Data Definition Language (DDDL)
  - DIDO Data Manipulation Language (DDML)
- CLI examples are not finalized and are not syntactically or semantically rigourous but provided as proof of concepts

DIDO CLI - Next Steps

- Work with a DIDO Platform vendor such as IOTA to refine the underlying CLI Data Model
- Submit the CLI Data Model for standardization
- Work with software language specialists to define a formal syntactically and semantically rigourous DDCL, DDDL and a DDML language
- Submit DDCL, DDDL and DDML CLIs for standardization
- Provide an Open Source Parser for the DIDO CLI that populates the CLI Data Model
- Provide a DDCL, DDDL and DDML Container for deployment on DIDO Nodes