DATALUTION: A TOOL FOR CONTINUOUS SCHEMA EVOLUTION IN NOSQL-BACKED WEB APPLICATIONS

STEFANIE SCHERZINGER
STEPHANIE SOMBAHL
KATHARINA WIECH
MEIKE KLETTKE
UTA STÖRL
Development Environment

development IDE

code repository

v0

commit
Development Environment

code repository

devlop IDE

commit

v0
Development Environment

code repository

development IDE

commit

v0

v1

Production Environment

PaaS

DaaS

v0

v0
The application code declares a schema.

The application code evolves.

Thus, we need to address schema evolution:

- Eager
- Lazy with Object-NoSQL Mappers
- Lazy with Datalution
EXAMPLE: GAMING APPLICATION

Release 1
• Player(ID, NAME)
• Mission(ID, TITLE, PID)

Release 2
• Players carry a property SCORE:
  add Player.SCORE = 50

Release 3
• Missions carry their player’s score
  copy Player.SCORE to Mission
  where Player.ID = Mission.PID
New release, changing the schema:
add Player.SCORE = 50

New release, changing the schema:
copy Player.SCORE to Mission where Player.ID = Mission.PID

Original schema imposed by the application:
Player(ID, NAME), Mission(ID, TITLE, PID)

Logical clock

EAGER MIGRATION
EAGER MIGRATION

Original schema imposed by the application:
Player(ID, NAME), Mission(ID, TITLE, PID)

New release, changing the schema:
add Player.SCORE = 50

Original schema imposed by the application:
Player(ID, NAME), Mission(ID, TITLE, PID)

New release, changing the schema:
copy Player.SCORE to Mission where Player.ID = Mission.PID

get(Mission, 100)

get(Mission, 101)

EAGER MIGRATION

Original schema imposed by the application:
Player(ID, NAME), Mission(ID, TITLE, PID)

New release, changing the schema:
add Player.SCORE = 50

Original schema imposed by the application:
Player(ID, NAME), Mission(ID, TITLE, PID)

New release, changing the schema:
copy Player.SCORE to Mission where Player.ID = Mission.PID

get(Mission, 100)

get(Mission, 101)

EAGER MIGRATION

Original schema imposed by the application:
Player(ID, NAME), Mission(ID, TITLE, PID)

New release, changing the schema:
add Player.SCORE = 50

Original schema imposed by the application:
Player(ID, NAME), Mission(ID, TITLE, PID)

New release, changing the schema:
copy Player.SCORE to Mission where Player.ID = Mission.PID

get(Mission, 100)

get(Mission, 101)

EAGER MIGRATION

Original schema imposed by the application:
Player(ID, NAME), Mission(ID, TITLE, PID)

New release, changing the schema:
add Player.SCORE = 50

Original schema imposed by the application:
Player(ID, NAME), Mission(ID, TITLE, PID)

New release, changing the schema:
copy Player.SCORE to Mission where Player.ID = Mission.PID

get(Mission, 100)

get(Mission, 101)

EAGER MIGRATION

Original schema imposed by the application:
Player(ID, NAME), Mission(ID, TITLE, PID)

New release, changing the schema:
add Player.SCORE = 50

Original schema imposed by the application:
Player(ID, NAME), Mission(ID, TITLE, PID)

New release, changing the schema:
copy Player.SCORE to Mission where Player.ID = Mission.PID

get(Mission, 100)

get(Mission, 101)

EAGER MIGRATION

Original schema imposed by the application:
Player(ID, NAME), Mission(ID, TITLE, PID)

New release, changing the schema:
add Player.SCORE = 50

Original schema imposed by the application:
Player(ID, NAME), Mission(ID, TITLE, PID)

New release, changing the schema:
copy Player.SCORE to Mission where Player.ID = Mission.PID

get(Mission, 100)

get(Mission, 101)

EAGER MIGRATION

Original schema imposed by the application:
Player(ID, NAME), Mission(ID, TITLE, PID)

New release, changing the schema:
add Player.SCORE = 50

Original schema imposed by the application:
Player(ID, NAME), Mission(ID, TITLE, PID)

New release, changing the schema:
copy Player.SCORE to Mission where Player.ID = Mission.PID

get(Mission, 100)

get(Mission, 101)

EAGER MIGRATION

Original schema imposed by the application:
Player(ID, NAME), Mission(ID, TITLE, PID)

New release, changing the schema:
add Player.SCORE = 50

Original schema imposed by the application:
Player(ID, NAME), Mission(ID, TITLE, PID)

New release, changing the schema:
copy Player.SCORE to Mission where Player.ID = Mission.PID

get(Mission, 100)

get(Mission, 101)

EAGER MIGRATION

Original schema imposed by the application:
Player(ID, NAME), Mission(ID, TITLE, PID)

New release, changing the schema:
add Player.SCORE = 50

Original schema imposed by the application:
Player(ID, NAME), Mission(ID, TITLE, PID)

New release, changing the schema:
copy Player.SCORE to Mission where Player.ID = Mission.PID

get(Mission, 100)

get(Mission, 101)

EAGER MIGRATION

Original schema imposed by the application:
Player(ID, NAME), Mission(ID, TITLE, PID)

New release, changing the schema:
add Player.SCORE = 50

Original schema imposed by the application:
Player(ID, NAME), Mission(ID, TITLE, PID)

New release, changing the schema:
copy Player.SCORE to Mission where Player.ID = Mission.PID

get(Mission, 100)

get(Mission, 101)

EAGER MIGRATION

Original schema imposed by the application:
Player(ID, NAME), Mission(ID, TITLE, PID)

New release, changing the schema:
add Player.SCORE = 50

Original schema imposed by the application:
Player(ID, NAME), Mission(ID, TITLE, PID)

New release, changing the schema:
copy Player.SCORE to Mission where Player.ID = Mission.PID

get(Mission, 100)

get(Mission, 101)

EAGER MIGRATION

Original schema imposed by the application:
Player(ID, NAME), Mission(ID, TITLE, PID)

New release, changing the schema:
add Player.SCORE = 50

Original schema imposed by the application:
Player(ID, NAME), Mission(ID, TITLE, PID)

New release, changing the schema:
copy Player.SCORE to Mission where Player.ID = Mission.PID

get(Mission, 100)

get(Mission, 101)

EAGER MIGRATION

Original schema imposed by the application:
Player(ID, NAME), Mission(ID, TITLE, PID)

New release, changing the schema:
add Player.SCORE = 50

Original schema imposed by the application:
Player(ID, NAME), Mission(ID, TITLE, PID)

New release, changing the schema:
copy Player.SCORE to Mission where Player.ID = Mission.PID

get(Mission, 100)

get(Mission, 101)

EAGER MIGRATION

Original schema imposed by the application:
Player(ID, NAME), Mission(ID, TITLE, PID)

New release, changing the schema:
add Player.SCORE = 50

Original schema imposed by the application:
Player(ID, NAME), Mission(ID, TITLE, PID)

New release, changing the schema:
copy Player.SCORE to Mission where Player.ID = Mission.PID

get(Mission, 100)

get(Mission, 101)
New release, changing the schema:
add Player.SCORE = 50

Original schema imposed by the application:
Player(ID, NAME),
Mission(ID, TITLE, PID)

New release, changing the schema:
copy Player.SCORE to Mission
where Player.ID = Mission.PID

get(Mission, 100)
get(Mission, 101)

Updating Lisa's player:
put (Player (1, "Lisa", 100))

EAGER MIGRATION

Logical clock

{"ID": 1,
 "NAME": "Lisa",
"ts": ts1}

{"ID": 2,
 "NAME": "Bart",
"ts": ts2}

{"ID": 3,
 "NAME": "Ralf",
"ts": ts3}

{"ID": 100,
 "TITLE": "tower",
"PID": 1,
"ts": ts4}

{"ID": 101,
 "TITLE": "manor",
"PID": 2,
"ts": ts5}

{"ID": 1,
 "NAME": "Lisa",
 "SCORE": 100,
 "ts": ts7}

{"ID": 1,
 "NAME": "Lisa",
 "SCORE": 50,
 "ts": ts6}

{"ID": 2,
 "NAME": "Bart",
 "SCORE": 50,
 "ts": ts6}

{"ID": 3,
 "NAME": "Ralf",
 "SCORE": 50,
 "ts": ts6}

Logical clock

"ts10"
New release, changing the schema:
add Player.SCORE = 50

copy Player.SCORE to Mission
where Player.ID = Mission.PID

Original schema imposed by the application:
Player(ID, NAME), Mission(ID, TITLE, PID)

EAGER MIGRATION

get(Mission, 100) get(Mission, 101)

Logical clock:
ts1 … ts5

{"ID": 1, "NAME": "Lisa", "ts": ts1}
{"ID": 2, "NAME": "Bart", "ts": ts2}
{"ID": 3, "NAME": "Ralf", "ts": ts3}
{"ID": 100, "TITLE": "tower", "PID": 1, "ts": ts4}
{"ID": 101, "TITLE": "manor", "PID": 2, "ts": ts5}

{"ID": 1, "NAME": "Lisa", "ts": ts6}
{"ID": 2, "NAME": "Bart", "SCORE": 50, "ts": ts7}
{"ID": 3, "NAME": "Ralf", "SCORE": 50, "ts": ts6}

{"ID": 1, "NAME": "Lisa", "ts": ts8}
{"ID": 100, "TITLE": "tower", "PID": 1, "SCORE": 100, "ts": ts9}
{"ID": 101, "TITLE": "manor", "PID": 2, "SCORE": 50, "ts": ts10}
New release, changing the schema:
add Player.SCORE = 50

copy Player.SCORE to Mission
where Player.ID = Mission.PID

Original schema imposed by the application:
Player(ID, NAME),
Mission(ID, TITLE, PID)

New release, changing the schema:
Updating Lisa's player:
put (Player (1, "Lisa", 100))

get(Mission, 100)

tget(Mission, 101)

logical clock

EAGER MIGRATION

Original schema imposed by the application:
Player(ID, NAME),
Mission(ID, TITLE, PID)

New release, changing the schema:
add Player.SCORE = 50

copy Player.SCORE to Mission
where Player.ID = Mission.PID

get(Mission, 100)

tget(Mission, 101)

logical clock
New release, changing the schema:
add Player.SCORE = 50

Original schema imposed by the application:
Player(ID, NAME), Mission(ID, TITLE, PID)

New release, changing the schema:
copy Player.SCORE to Mission
where Player.ID = Mission.PID

get(Mission, 100)
get(Mission, 101)

Updating Lisa's player:
put (Player (1, "Lisa", 100))

get(Mission, 100)
get(Mission, 101)

Logical clock

EAGER MIGRATION
LAZY EVOLUTION WITH OBJECT-NOSQL MAPPERS

Original schema imposed by the application:

```java
@Entity
class Player{
    @Id
    Integer ID;

    String NAME;
    ...
}
```

New release, changing the schema:

```java
@Entity
class Player{
    @Id
    Integer ID;

    String NAME;
    Integer SCORE = 50;
}
```

Convenient "quick fix" for simple changes.

Long-term: Maintenance nightmare.

put(…)

get(Player, 1)
LAZY MIGRATION IN DATALUTION

Original schema imposed by the application:
Player(ID, NAME),
Mission(ID, TITLE, PID)

New release, changing the schema:
add Player.SCORE = 50

Updating Lisa’s player:
put (Player (1, “Lisa”, 100))

New release, changing the schema:
copy Player.SCORE to Mission
where Player.ID = Mission.PID

get(Mission, 100)
get(Mission, 101)

ts1 … ts5
ts6
	Player
{“ID”: 1,
“NAME”: “Lisa”,
“ts”: ts1}

Player
{“ID”: 2,
“NAME”: “Bart”,
“ts”: ts2}

Player
{“ID”: 3,
“NAME”: “Ralf”,
“ts”: ts3}

Mission
{“ID”: 100,
“TITLE”: “tower”,
“PID”: 1,
“ts”: ts4}

Mission
{“ID”: 101,
“TITLE”: “manor”,
“PID”: 2,
“ts”: ts5}

ts7

ts8
ts9
ts10
logical clock
LAZY MIGRATION IN DATALUTION

Original schema imposed by the application:
Player(ID, NAME),
Mission(ID, TITLE, PID)

New release, changing the schema:
add Player.SCORE = 50

New release, changing the schema:
copy Player.SCORE to Mission
where Player.ID = Mission.PID

Original schema imposed by the application:
Player(ID, NAME),
Mission(ID, TITLE, PID)

Updating Lisa's player:
put (Player (1, "Lisa", 100))

New release, changing the schema:
copy Player.SCORE to Mission
where Player.ID = Mission.PID

get(Mission, 100)
get(Mission, 101)

get(Mission, 100)
get(Mission, 101)

Logical clock
LAZY MIGRATION IN DATALUTION

Original schema imposed by the application:
Player(ID, NAME),
Mission(ID, TITLE, PID)

New release, changing the schema:
add Player.SCORE = 50

New release, changing the schema:
copy Player.SCORE to Mission
where Player.ID = Mission.PID

Original schema imposed by the application:
Player(ID, NAME),
Mission(ID, TITLE, PID)

New release, changing the schema:
add Player.SCORE = 50

New release, changing the schema:
copy Player.SCORE to Mission
where Player.ID = Mission.PID

get(Mission, 100)
get(Mission, 101)
LAZY MIGRATION
IN DATALUTION
New release, changing the schema:
add Player.SCORE = 50

New release, changing the schema:
copy Player.SCORE to Mission
where Player.ID = Mission.PID

Original schema imposed by
the application:
Player(ID, NAME),
Mission(ID, TITLE, PID)

get(Mission, 100)
get(Mission, 101)

Updating Lisa’s player:
put (Player (1, “Lisa”, 100))

LAZY MIGRATION
IN DATALUTION
LAZY MIGRATION IN DATALUTION

Original schema imposed by the application:
Player(ID, NAME),
Mission(ID, TITLE, PID)

New release, changing the schema:
add Player.SCORE = 50

Updating Lisa's player:
put (Player (1, "Lisa", 100))

New release, changing the schema:
copy Player.SCORE to Mission
where Player.ID = Mission.PID

get(Mission, 100)
get(Mission, 101)

LAZY MIGRATION
IN DATALUTION

complete lazy migration

logical clock
DATALOG MODEL
(NONRECURSIVE, STRATIFIED)

a1: put(Player(1, "Lisa"));
a2: put(Player(1, "Lisa S."));

    r1: Player(1, "Lisa", ts1).
    r2: Player(1, "Lisa S.", ts2).

a3: get("Player", 1);

    r3: legacyPlayer(ID, TS) :-
        Player(ID, _, TS), Player(ID, _, NTS), TS < NTS.

    r4: latestPlayer(ID, TS) :-
        Player(ID, _, TS), not legacyPlayer(ID, TS).

    r5: getPlayer(ID, NAME, TS) :-
        Player(ID, NAME, TS), latestPlayer(ID, TS).

transient rule – derived facts not kept around for incremental evaluation
Let \( \text{kind}[r](ID, P_1, ..., P_n) \) be the schema imposed by the current application release. \( ts \) denotes a fresh timestamp associated with release \( r \).

i) **add**: \( \text{kind}.P_{n+1} = v \), where \( P_{n+1} \) is a new property name and \( v \) is a default value (in the new version of the entity, \( P_{n+1} \) has value \( v \)): 

\[
\text{kind}[r+1](ID, P_1, ..., P_n, v, ts) \iff \text{kind}[r](ID, P_1, ..., P_n, \text{OTS}), \text{latest\text{kind}}[r](ID, \text{OTS}).
\]

ii) **delete**: \( \text{kind}.P_i \)

\[
\text{kind}[r+1](ID, P_1, ..., P(i-1), P(i+1), ..., P_n, ts) \iff \text{kind}[r](ID, P_1, ..., P_n, \text{OTS}), \text{latest\text{kind}}[r](ID, \text{OTS}).
\]

Let \( \text{kindS}[r](ID, S_1, ..., S_n) \) and \( \text{kindT}[r](ID, T_1, ..., T_m) \) be the current source and target schema imposed by the application.

iii) **copy**: \( \text{kindS}.S_i \) to \( \text{kindT} \) where \( \text{kindS}.ID = \text{kindT}.T_j \)

\[
\text{kindT}[r+1](ID_T, T_1, ..., T_m, S_i, ts) \iff \text{kindT}[r](ID_T, T_1, ..., T_m, \text{TS_T}), \text{latest\text{kindT}}[r](ID_T, \text{TS_T}), \text{kindS}[r](ID_S, S_1, ..., S_n, \text{TS_S}), \text{latest\text{kindS}}[r](ID_S, \text{TS_S}), \text{ID_S} = T_j.
\]

\[
\text{kindT}[r+1](ID_T, T_1, ..., T_m, \text{null}, ts) \iff \text{kindT}[r](ID_T, T_1, ..., T_m, \text{TS_T}), \text{latest\text{kindT}}[r](ID_T, \text{TS_T}), \not \text{kindS}[r](ID_S, S_1, ..., S_n, \text{TS_S}), \text{ID_S} = T_j.
\]

\[
\text{kindS}[r+1](ID, S_1, ..., S_n, ts) \iff \text{kindS}[r](ID, S_1, ..., S_n, \text{OTS}), \text{latest\text{kind}}[r](ID, \text{OTS}).
\]

iv) **move**: \( \text{kindS}.S_i \) to \( \text{kindT} \) where \( \text{kindS}.ID = \text{kindT}.T_j \), with the same first two rules as for copy, as well as the following rule:

\[
\text{kindS}[r+1](ID, S_1, ..., S(i-1), S(i+1), ..., S_n, ts) \iff \text{kindS}[r](ID, S_1, ..., S_n, \text{OTS}), \text{latest\text{kindS}}[r](ID, \text{OTS}).
\]
DATALUTION: DATALOG-BASED

- Eager migration: Incremental bottom-up evaluation
- Lazy migration: Incremental top-down evaluation
  - Employing sideways information passing strategies
  - Exploiting uniqueness of identifiers

- Both strategies always yield the same result

- Progress:
  - Theory in DBPL@SPLASH'15 paper
  - Demo of PoC Datalution at QUDOS'16
  - Ongoing: Integration with NoSQL data store