Links: A Tierless Web Programming Language Providing Language Integrated Query (now with Added Time Travel)

Vashti Galpin Frank Emrich

School of Informatics
University of Edinburgh

SPLV 2023
24 July 2023

THE UNIVERSITY of EDINBURGH
informatics

Links: A Tierless Web Programming Language with LINQ

Outline

- → Links as a programming language, example Links web app
- → Three case studies
- → Temporal features of Links (time travel)

Terminology

- → LINQ: language integrated query
- → SQL: relational database programming language
- → DOM: Document Object Model, HTML in web browsers
- → Provenance: how a piece of data has been created/modified
- → Digital/data curation: management and preservation of data

Logistics

- → Please ask questions at any time (including after the talk)
- → Pointers to resources at the end of the slides
- → Slides available at
  `https://github.com/links-lang/splv2023_example_code`

Links: The Team

Then: Phil Wadler, Jeremy Wallop, Sam Lindley, Ezra Cooper

Inbetween: many PhD students, interns, visitors, postdocs and researchers who are now doing other things

Now: Sam Lindley, Phil Wadler and

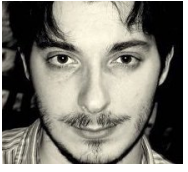
James Cheney

Frank Emrich

Simon Fowler

Vashti Galpin

Daniel Hillerström



Wilmer Ricciotti

Links: Timeline Part I

2006 Microsoft LINQ, C# and F#, reflection and quotation
[Meier et al (SIGMOD 2006), Syme (2006)]

2006 Development of Links based on Nested Relational Calculus and Kleisli
[Cooper et al (FMCO 2006), Buneman et al (TCS 1995), Wong (JFP 2000)]

2008 Formlets: abstraction for HTML forms [Cooper et al (APLAS 2008)]

2009 Use of effect typing for LINQ, normalisation of high-order NRC queries
[Cooper (DBPL 2009)]

2009 RPC calculus, location-aware programming
[Cooper & Wadler (PPDP 2009), Choi et al (SCP 2020, PPDP 2021)]

2012 Row-based effect types for database integration [Lindley & Cheney (TLDI 2012)]

2013 Theory of language-integrated query based on quotation
[Cheney et al (ICFP 2013, PEPM 2014)]

2014 Query shredding: efficient nested multiset queries
[Cheney et al (SIGMOD 2014)]

Links: Timeline Part II

2016 Effect handlers
[Hillerström et al (TyDe 2016, FSCD 2017, APLAS 2018, JFP 2020, ICFP 2020)]

2016 Provenance [Fehrenbach & Cheney (PPDP 2016, SCP 2018, DBPL 2019)]

2017 Session types, concurrency & distribution
[Lindley and Morris (2017), Fowler et al (POPL 2019)]

2018 Relational lenses, view update [Horn et al. (ICFP 2018, IFL 2019)]

2019 Model-View-Update using session types [Fowler (ECOOP 2020)]

2021 Temporal database features: transaction-time, valid-time
[Cheney and Galpin (IPAW 2021), Fowler et al (GCPE 2022)]

2021 Semantics for null values, rewrite rules for normalising to SQL
[Cheney & Ricciotti (DBPL 2021), Ricciotti & Cheney (JAR 2022)]

2021 Set and multiset semantics, deduplication and multiset difference
[Ricciotti & Cheney (ESOP 2021, LMCS 2022)]

2023 Queries over finite maps, grouping and aggregation [Ricciotti (PPDP 2023)]

Links: A Tierless Web Programming Language

Links in a Nutshell

The language

- → Functional
- → Expression-based
- → Syntax loosely inspired by JavaScript
- → Strongly typed, statically typed

Type system

- → Type inference
- → Structural instead of nominal types
- → Row polymorphism
- → Effect typing, linearity, session types, first-class polymorphism, …

Structural typing in Links

```
# A record type containing fields id and descr
typename Task = (id: Int, descr: String);

fun makeTask(id : Int, descr: String) {
  (id=id, descr=descr)
}
```

Inferred return type of `makeTask`: `(id: Int, descr: String)`

→ Equivalent to `Task`

→ … but definition of `Task` irrelevant for typing `makeTask`

Omitting type annotations yields more general type (slightly simplified):
$\forall a, b.(a, b) \rightarrow (\texttt{id} : a, \texttt{descr} : b)$

A Glimpse at Row Polymorphism, Part I

```
typename Task = (id: Int, descr: String);

fun maxId(tasks) {
  switch (tasks) {
    case [] -> -1
    case task :: xs -> maximum(task.id, maxId(xs))
  }
}
```

What should the type of `tasks` be? Hint: Not `[Task]`!

All we learn: elements of `tasks` are records with **at least** field `id : Int`

Row variables ρ abstract over "unknown" parts of record.

Inferred type (slightly simplified): $\forall \rho : \texttt{Row}.([(\texttt{id} : \texttt{Int} | \rho)]) \to \texttt{Int}$

A Glimpse at Row Polymorphism, Part II

Inferred type of `maxId` (slightly simplified): $\forall \rho : \texttt{Row}.([(\texttt{id} : \texttt{Int} | \rho)]) \to \texttt{Int}$

As usual: Instantiate quantified variables to obtain different types for `maxId`

Variable ρ is instantiated with rows rather than ordinary types

- $\to$ Choosing ρ to be the empty row:
  $([(\texttt{id} : \texttt{Int})]) \to \texttt{Int}$
- $\to$ Choosing ρ to be row ($\texttt{descr} : \texttt{String}$):
  $([(\texttt{id} : \texttt{Int}, \texttt{descr} : \texttt{String})]) \to \texttt{Int}$
- $\to$ Choosing ρ to be row with fields other than `id`:
  $([(\texttt{id} : \texttt{Int}, \texttt{first\_name} : \texttt{String}, \texttt{family\_name} : \texttt{String})]) \to \texttt{Int}$

Effect Typing

Links' type system tracks what **side-effects** a function may have. There is an effect that prevents impure functions from being part of query comprehensions.

Function types (in the context of this talk and slightly simplified):

- `(T1, ..., Tn) -> T` - pure, without side effects or recursion
- `(T1, ..., Tn) ~> T` - impure, may have side effects or recursion

```
fun impureFunction() {
  print("hello");
  3
}
# type function into REPL and then enter
links> impureFunction;
fun : () ~> Int
```

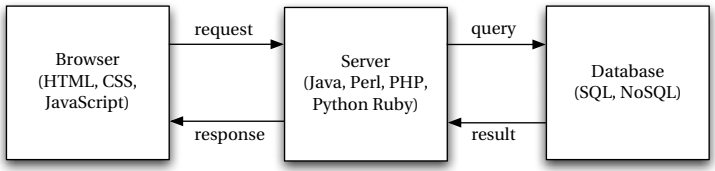
Links also supports user-defined effects through **effect handlers**, reflected in types

Links: A Tierless Web Programming Language

Tierless Web Programming

Web application tiers

- → Server
- → Client (= Browser)
- → Database

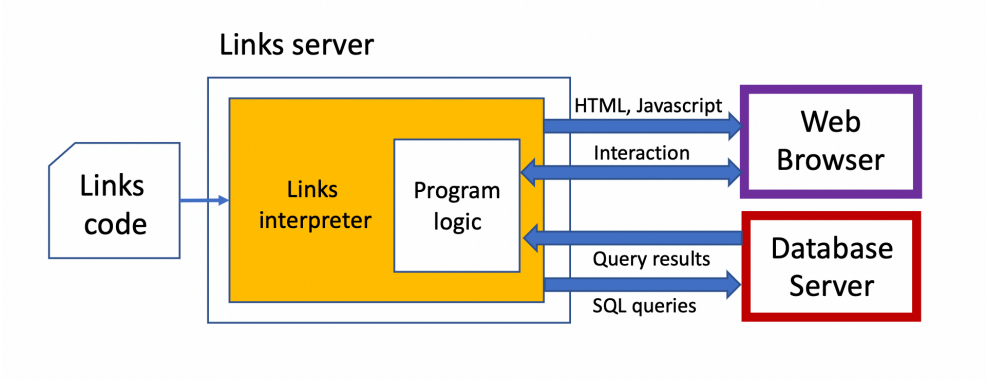


Tierless web programming

- → Single Links program contains application logic for all three tiers
- → HTML fragments can be embedded directly
- → Language-integrated database queries expressed using comprehension syntax

Execution

Translation to SQL and JavaScript automatically performed by Links



Communication between server and client(s):

→ Implicitly using remote procedure calls (RPCs)

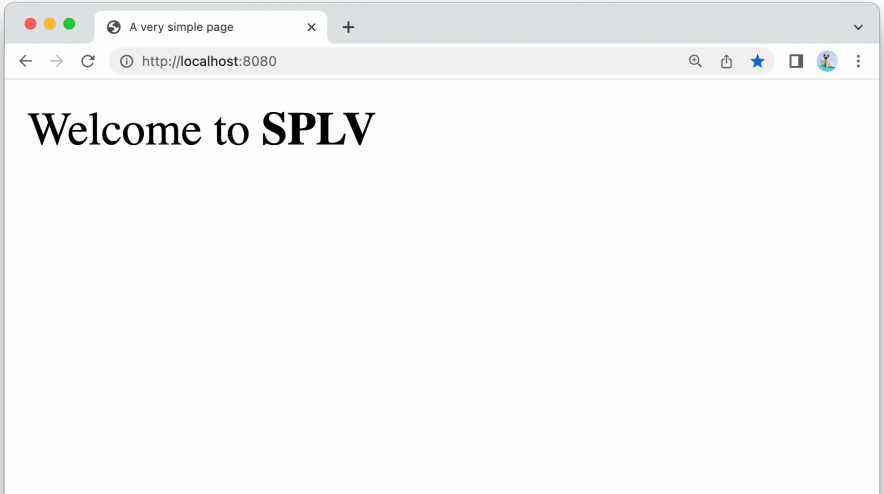
→ Explicitly by sending messages

Serving a Very Simple Page

```
fun createPage(event) {
  page
    <html>
      <title>A very simple page</title>
      <body>
        Welcome to {event}
      </body>
    </html>
}

addRoute("", fun(_) {createPage(<b>SPLV</b>)});
servePages()
```

A Very Simple Page

```
> linx --config=config simple.links
```



Simple DOM Manipulations

Links provides builtin functions to manipulate DOM tree

```
sig addTaskToDom : (String) ~> ()
fun addTaskToDom(task) client {
  # Assumes that there is a <table> element
  # with id "table"
  var t = getNodeById("table");
  var new_row =
      <tr><td>{stringToXml(task)}</td></tr>;
  appendChildren(new_row, t)
}
```

Note the annotation `client` on `addTaskToDom`!

Locations

Specification

→ May annotate function f with location l $\in$ `client`, `server`
→ “f only exists on l”
→ Functions without an annotation exist on both

Behaviour

→ Calling function not present on current location results in RPC
→ Automatic (de)serialisation

Constraints

→ DOM manipulation only allowed on client
→ Issuing database queries only allowed on server
→ Not enforced by type system!

Forms

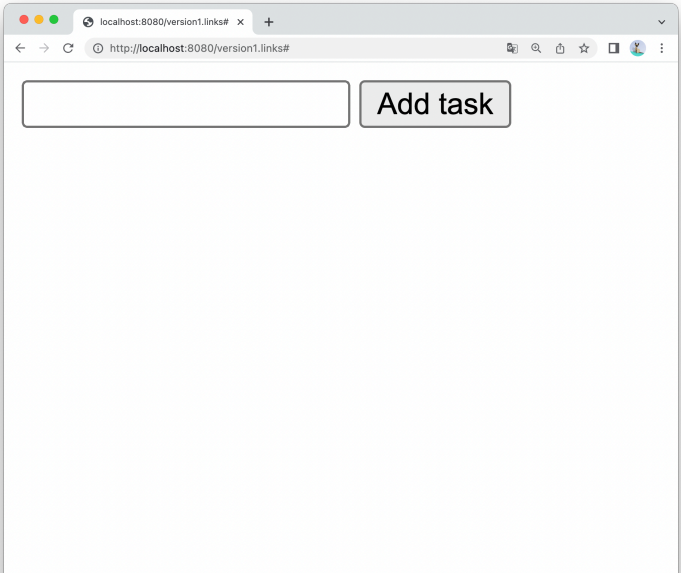
```
fun createPage() {
  page
    <html><body>
        <form l:onsubmit="{addTaskToDom( task )}">
          <input l:name=" task "/>
          <button type="submit">Add task</button>
        </form>
        <table id="table"></table>
    </body></html>
}
```

Simple mechanism for data input

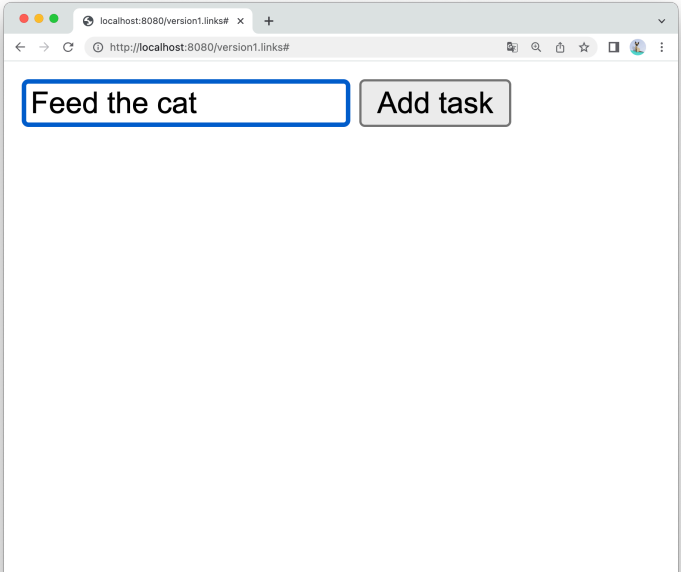
→ `l:onsubmit` handler expects expression of type unit

→ `task` in scope!

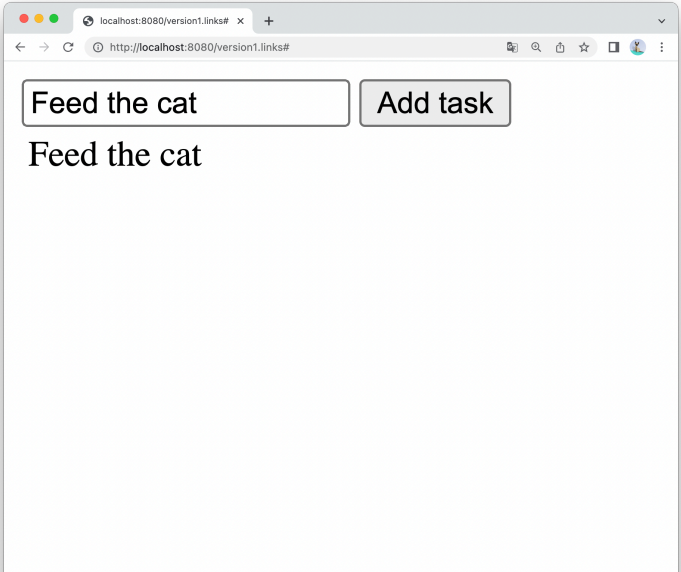
A Simple Web App



A Simple Web App



A Simple Web App



Managing State

Previous example

→ No explicit representation of program state, only implicitly in DOM tree
→ Not suitable for real-world application

Towards explicit state

```
typename Task = String;
typename State = [Task];

sig makeTable : (State) -> Xml # creates <table>
fun makeTable(tasks) {...}

sig makeBody : (State) ~> Xml # creates <body>
fun makeBody(tasks) {...}
```

Full Example

```
sig makeTable : (State) -> Xml
fun makeTable(tasks) {...}

mutual {
 sig redraw : (State) ~> ()
 fun redraw(tasks) client {...}

 sig addTask : (State, Task) ~> ()
 fun addTask(tasks, task) {...}

 sig makeBody : (State) ~> Xml
 fun makeBody(tasks) {...}
}

fun createPage() {...}
```

Full Example

```
sig makeTable : (State) -> Xml
fun makeTable(tasks) {...}

mutual {
 sig redraw : (State) ~> ()
 fun redraw(tasks) client {...}

 sig addTask : (State, Task) ~> ()
 fun addTask(tasks, task) {...}

 sig makeBody : (State) ~> Xml
 fun makeBody(tasks) {...}
}

fun createPage() {...}
```

In detail:

```
fun createPage() {
  page
   <html>
     <body>
       {makeBody([])}
     </body>
   </html>
}
```

Full Example

```
sig makeTable : (State) -> Xml
fun makeTable(tasks) {...}

mutual {
 sig redraw : (State) ~> ()
 fun redraw(tasks) client {...}

 sig addTask : (State, Task) ~> ()
 fun addTask(tasks, task) {...}

 sig makeBody : (State) ~> Xml
 fun makeBody(tasks) {...}
}

fun createPage() {...}
```

In detail:

```
fun makeBody(ts) {
 <body id="body">
  <form
    l:onsubmit="{addTask(ts, task)}">
   <input l:name="task"/>
   <button type="submit">
    Add task
   </button>
  </form>
  {makeTable(tasks)}
 </body>
}
```

Full Example

```
sig makeTable : (State) -> Xml
fun makeTable(tasks) {...}

mutual {
 sig redraw : (State) ~> ()
 fun redraw(tasks) client {...}

 sig addTask : (State, Task) ~> ()
 fun addTask(tasks, task) {...}

 sig makeBody : (State) ~> Xml
 fun makeBody(tasks) {...}
}

fun createPage() {...}
```

In detail:

```
fun makeTable(tasks) {
  <table id="table">
   {for (task <- tasks)
     <tr>
      <td>
       {stringToXml(task)}
      </td>
     </tr>
   }
  </table>
}
```

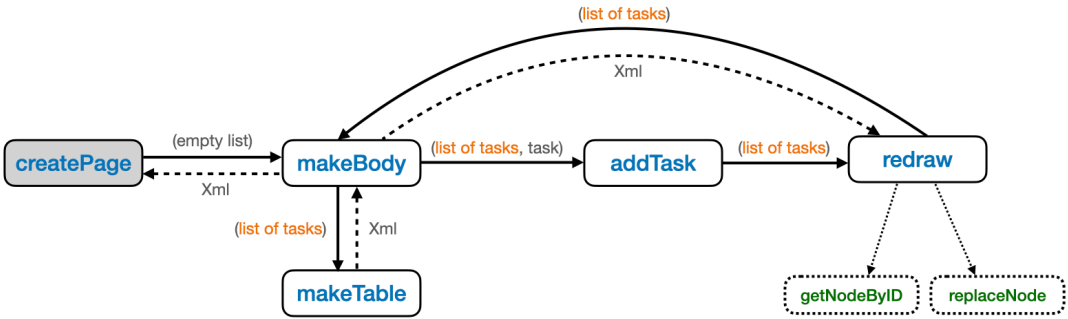
Full Example

```
sig makeTable : (State) -> Xml
fun makeTable(tasks) {...}

mutual {
 sig redraw : (State) ~> ()
 fun redraw(tasks) client {...}

 sig addTask : (State, Task) ~> ()
 fun addTask(tasks, task) {...}

 sig makeBody : (State) ~> Xml
 fun makeBody(tasks) {...}
}

fun createPage() {...}
```

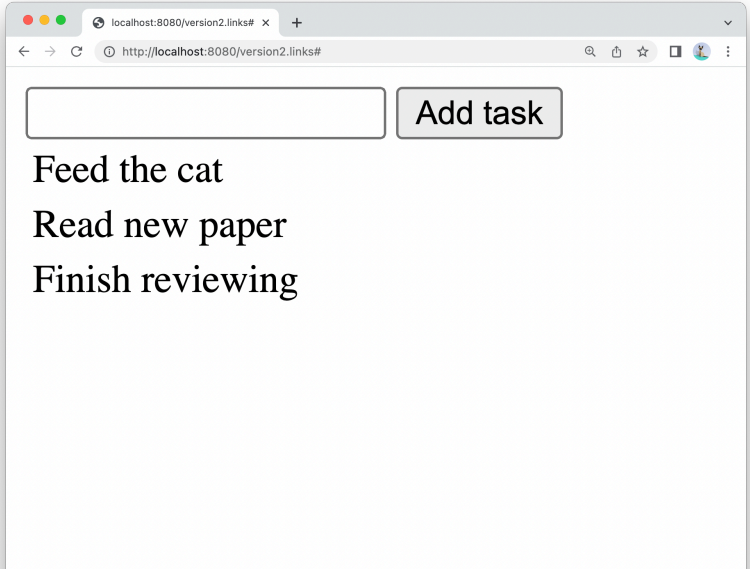
In detail:

```
fun addTask(tasks, descr) {
  var new_tasks = tasks ++ [descr];
  redraw(new_tasks)
}

fun redraw(tasks) client {
  var old_body = getNodeById("body");
  var new_body = makeBody(tasks);
  replaceNode(new_body,old_body)
}
```

A Simple Web App



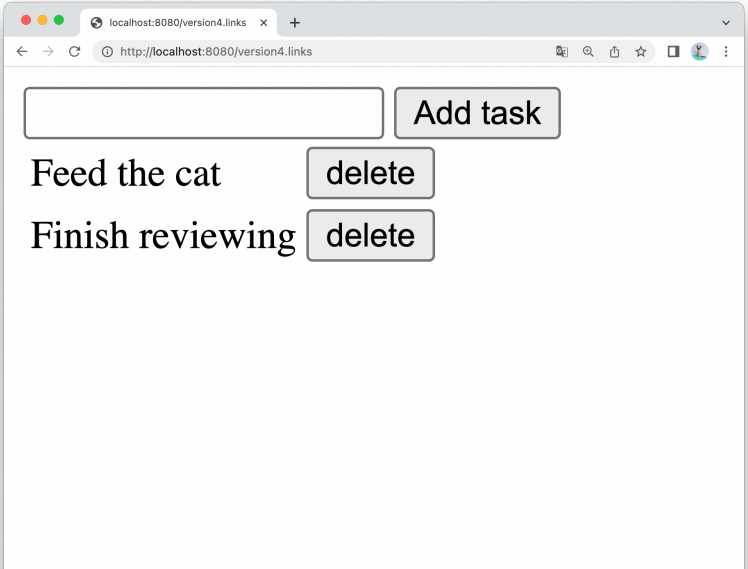
A Simple Web App



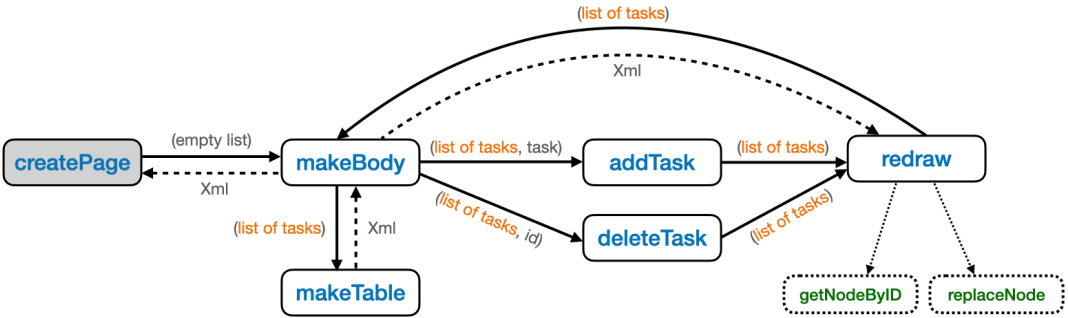
Supporting Deletion

```
typename Task = (id: Int, descr: String);
...
fun deleteTask(tasks, delete_id) {
  var new_tasks = filter(fun(t) t.id <> delete_id, tasks);
  redraw(new_tasks)/
}
...
fun makeTable(tasks) {
 <table id="table">
  {for (task <- tasks)
   <tr>
     ...
    <td><button l:onclick="{deleteTask(tasks, task.id)}" ... </td>
   </tr>}
 </table>
}
```

A Simple Web App



A Simple Web App



Persisting State

Language-integrated query: write DB queries in host language

- → Static typechecking of queries and results
- → Avoids security issues caused by building queries from strings
- → Potential performance gains from combining queries (see GtoPdb implementation)

```
SELECT * FROM todo
  WHERE descr = 'Eat'
```

```
query { for (x <-- todo)
  where (x.descr == "Eat")
    [x]}
```

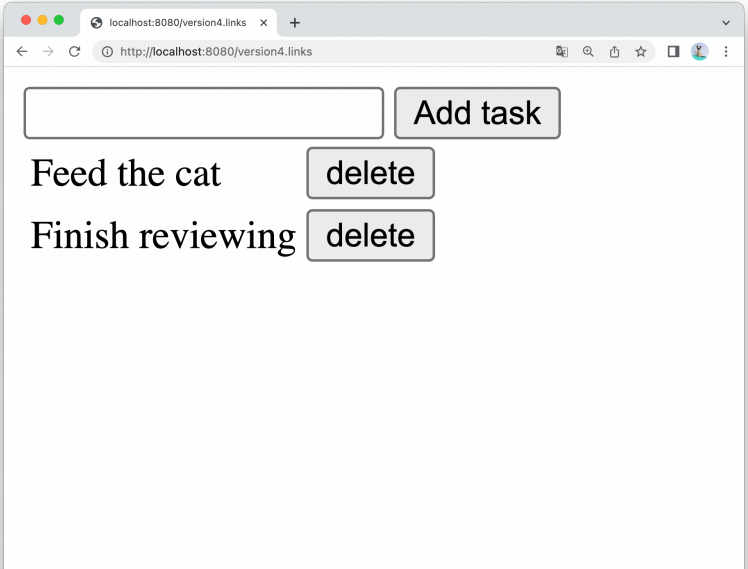
DB Support for Example Application: Loading

```
var db = database "tasks.db";
var tasks_db = table "tasks" with (id : Int, descr : String) from db;

fun loadTasks() {
    query {for (task <-- tasks_db) [task] }
}

...

addRoute("", fun(_) {createPage(loadTasks())});
```

DB Support for Example Application: Updating

```
...

fun addTask(tasks, descr) server {
  var id = maxId(tasks) + 1;
  var new_task = (id=id, descr=descr);
  insert (tasks_db) values [(id=new_task.id, descr=new_task.descr)];
  var new_tasks = tasks ++ [new_task];
  redraw(new_tasks)
}

fun deleteTask(tasks, delete_id) server {
  var new_tasks = filter(fun(t) {t.id <> delete_id}, tasks);
  delete (t <-- tasks_db) where (t.id == delete_id);
  redraw(new_tasks)
}
```

A Simple Web App



Pause for Reflection

Issues with example application

- → State manually passed around
- → Manual modifications to DOM tree, overriding whole `<body>`
- → Application logic, database access and presentation mixed

More desirable solution

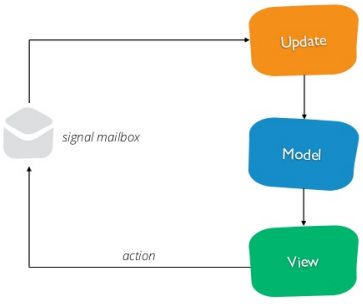
- → Clear separation of state, logic for changing it, and presentation
- → Perform database updates asynchronously

The building blocks (asynchronous processes, communication via messages) are present in Links…

… but so is a better solution!

Model-View-Update

Approach towards design of interactive applications in functional style, originally introduced by Elm language



Model: State of application
View: Renders model as HTML
Update: Updates model based on UI messages

Recently implemented in Links by Jake Browning and Simon Fowler

(MVU figure: https://www.slideshare.net/RogerioChaves1/introduction-to-elm)

MVU framework

```
# Model-view-update (MVU) GUI paradigm: separating state from HTML rendering

   open import  Mvu; open import ...;                          # import MVU and other libraries

   typename ...
   typename Model = ( results: [ResultType], ... )             # define model and messages types
   typename Messages = [| Continue | GetCat:String | ShowData | ... |];

   fun ...                                                     # program logic

   sig view : (Model) ~> HTML(Message)                         # define a function to render
   fun view(model) {                                           # the webpage
      <functions to generate HTML and messages>
      <body includes calls to functions that access database>
   }

   sig updt : (Message, Model) {}~> Model                      # define a function to respond
   fun updt(msg, model) {                                      # to messages
     switch(msg) {
        case Continue   -> redirect("/dataOverview"); model
        case GetCat(g)  -> (model with displayCat=g)
        case ShowData   -> lookupData(model)
        case ... -> ...
     }
   }
```

Case study: The Guide to Pharmacology

Curated Scientific Databases

Curated Databases:

→ “databases that are populated and updated with a great deal of human effort” (Buneman et al., 2008)

→ Examples: CIA World Factbook, GtoPdb

Curation challenges:

→ Annotation propagation, provenance tracking, citation, archiving…

→ Have to be hand-crafted for every curated DB — often with very stretched resources

Goal: programming language support for curation functionality

GtoPdb: IUPHAR/BPS Guide to PHARMACOLOGY

Popular curated database detailing around 3000 pharmacological targets (e.g., receptors) and 12000 ligands (e.g., drugs)

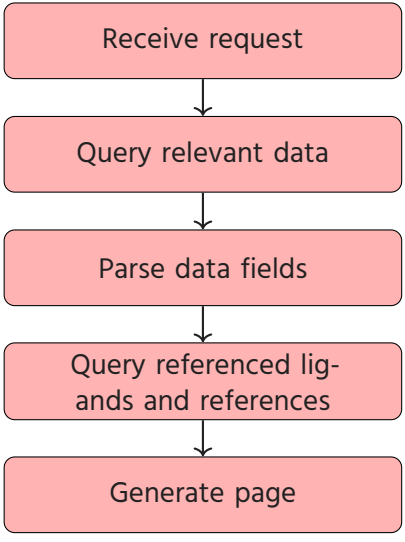
Initially IUPHAR-DB, started in 2003

Manually updated by 2-3 full-time curation staff at UoE, on advice of subcommittees worldwide

Written in Java. Scale (as of late 2019):

- → **89MB** (now 282MB) of data in **181 tables**
- → **17935 lines** of data transformation code
- → **28819 lines** of JSP rendering code
- → **43129 lines** of data access code

GtoPdb Page Lifecycle




```
Some substituted benzazepines such as SKF-83959
are G-protein biased agonists of the dopamine
D<sub>1</sub> receptor and fail to activate
&beta;-arrestin recruitment <Reference id=28036/>
their ability to signal through
G<sub>q</sub>-mediated pathways has been
controversial <Reference id=33435/>.
```

⇓

Some substituted benzazepines such as SKF-83959 are G-protein biased agonists of the dopamine D_1 receptor and fail to activate β-arrestin recruitment [17]; their ability to signal through G_q-mediated pathways has been controversial [32].

Nested Queries

```
for ( l <-- ligand )
  [( name = l.name,
     synonyms =
       for ( l2s <-- ligand2synonym )
         where ( l2s.ligand_id == l.ligand_id )
         [ l2s.synonym ]
  )]
```

```
[(name=L1,synonyms=[S1,S2]),(name=L2,synonyms=[]),...]
```

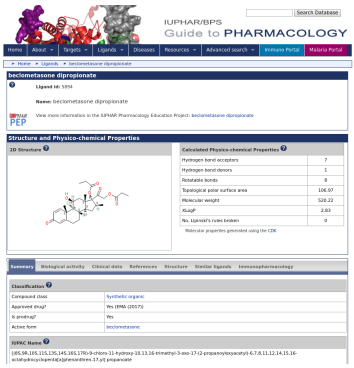
→ Links queries can be nested (unlike SQL queries which are always flat)

→ Query shredding technique provides a **static bound on query count**

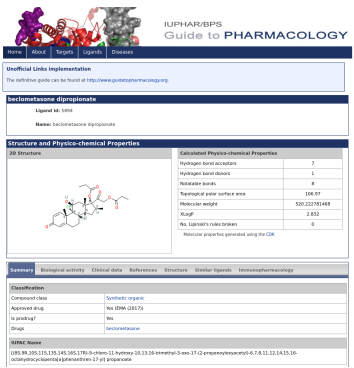
→ Nested queries are used **extensively** in GtoPdb, as shown earlier

Reimplentation in Links

Official GtoPdb



Links Reimplementation



All major data pages reimplemented

10981 lines of Links code, about 4 months of work

Underlying database unchanged

Evaluation

Query count

→ Number of queries performed per page load

→ **Expectation**: Links lower due to static query bounds ✓

Query handling time

→ Time taken to execute query and process results into data structures

→ **Expectation**: Either Links performs better due to fewer queries, or Java performs better due to faster marshalling ✓

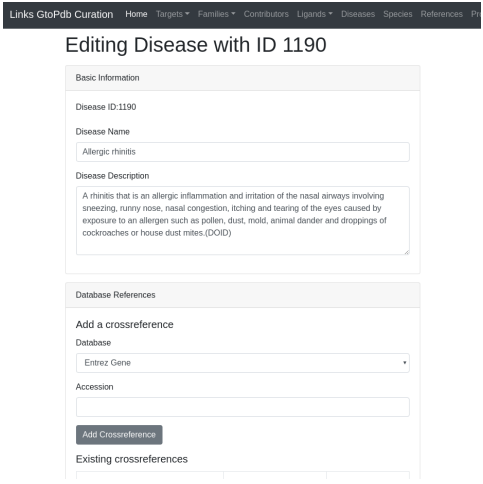
Page build time

→ Time between request handler invocation and response being ready

→ **Expectation**: Java to perform better due to JIT and maturity ✓

Performance measurements performed locally, using an instrumented Links interpreter and modified GtoPdb code. Sample of 150 random pages from object data and disease data pages.

Ongoing Work: Curation Interface



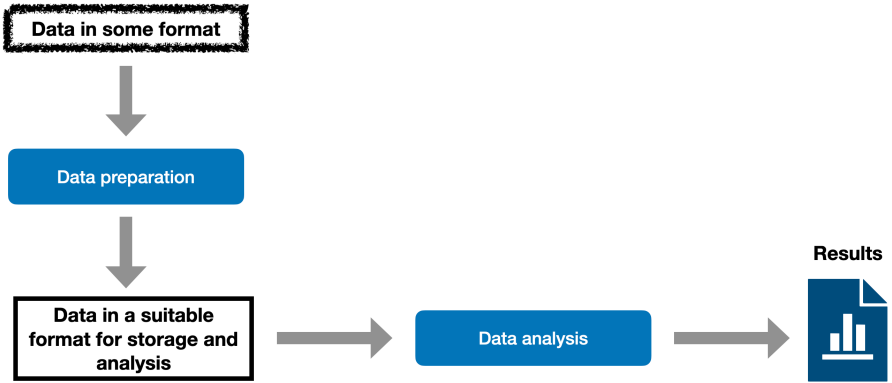
Ongoing work: Curation interface

→ Implemented two illustrative pages from curation interface

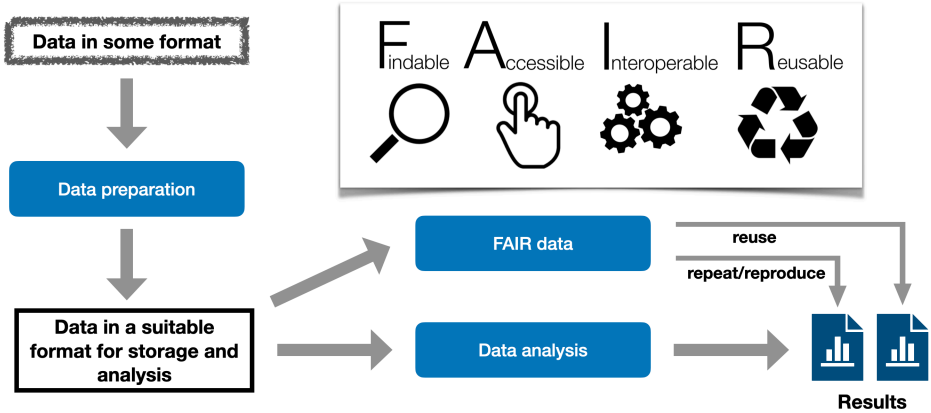
→ Aim to use these as realistic case study for curation functionality

Temporal case study: Covid 19 Data

Scientific workflow



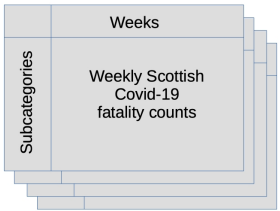
FAIR Scientific workflow



Making public data usable

National Records of Scotland

www.nrscotland.gov.uk/statistics-and-data



CSV files

Publicly available data

→ Released weekly
→ CSV files
→ Columns are labelled with weeks
→ Rows are labelled with subcategories
Category: Age
Subcategories: Under 1, 01-14, 15-44, 45-64, 65-74, 75-84, 85+

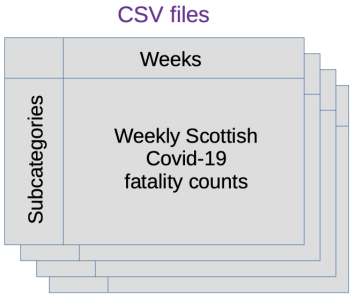
Build a web-front end

→ Use Links for rapid development

Sample data

Table 1: Weekly …							
Week number		1	…	11	12	13	…
Week beginning		30-Dec-19	…	9-Mar-20	16-Mar-20	23-Mar-20	…
Deaths		0	…	0	10	62	…
Deaths - females		0	…	0	5	26	…
Deaths - males		0	…	0	5	36	…
	Deaths by age						
	Under 1 year	0	…	0	0	0	…
	01-14	0	…	0	0	0	…
	15-44	0	…	0	0	0	…
	45-64	0	…	0	1	12	…
	65-74	0	…	0	4	11	…
	85+	0	…	0	2	15	…
…	…	…	…	…	…	…	…

CSV to relational database



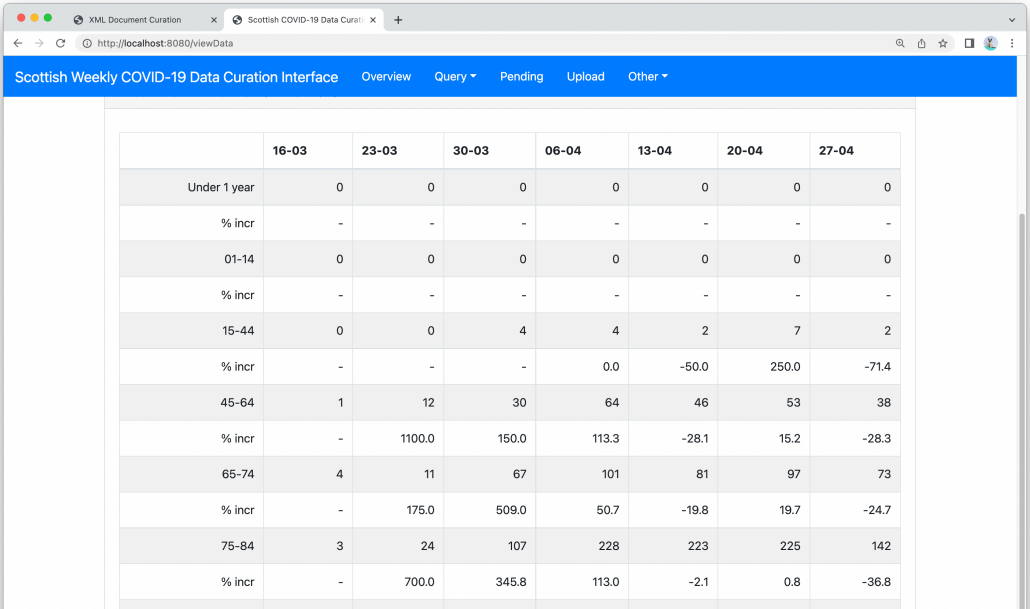
National Records of Scotland

www.nrscotland.gov.uk/statistics-and-data

Temporal database table

Week date	Sub-category	Fatality count	File_id

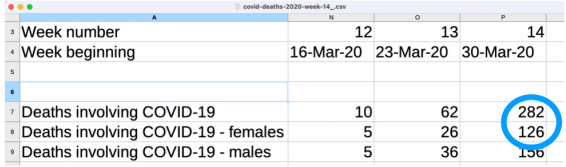
Data comparison



	16-03	23-03	30-03	06-04	13-04	20-04	27-04
Under 1 year	0	0	0	0	0	0	0
% incr	-	-	-	-	-	-	-
01-14	0	0	0	0	0	0	0
% incr	-	-	-	-	-	-	-
15-44	0	0	4	4	2	7	2
% incr	-	-	-	0.0	-50.0	250.0	-71.4
45-64	1	12	30	64	46	53	38
% incr	-	1100.0	150.0	113.3	-28.1	15.2	-28.3
65-74	4	11	67	101	81	97	73
% incr	-	175.0	509.0	50.7	-19.8	19.7	-24.7
75-84	3	24	107	228	223	225	142
% incr	-	700.0	345.8	113.0	-2.1	0.8	-36.8

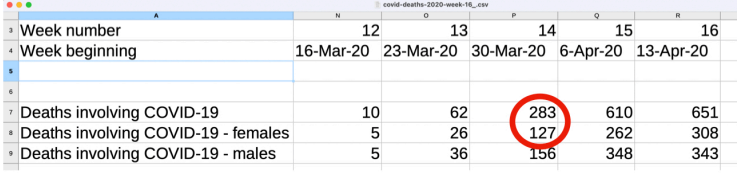
Data contains updates!

New data from the CSV of 30-Mar-2020


covid-deaths-2020-week-14_.csv

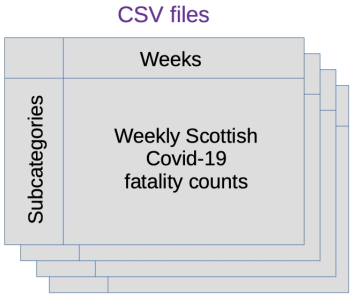
	A	N	O	P
3	Week number	12	13	14
4	Week beginning	16-Mar-20	23-Mar-20	30-Mar-20
5				
6				
7	Deaths involving COVID-19	10	62	282
8	Deaths involving COVID-19 - females	5	26	126
9	Deaths involving COVID-19 - males	5	36	156

Updates from the CSV of 13-Apr-2020


covid-deaths-2020-week-16_.csv

	A	N	O	P	Q	R
3	Week number	12	13	14	15	16
4	Week beginning	16-Mar-20	23-Mar-20	30-Mar-20	6-Apr-20	13-Apr-20
5						
6						
7	Deaths involving COVID-19	10	62	283	610	651
8	Deaths involving COVID-19 - females	5	26	127	262	308
9	Deaths involving COVID-19 - males	5	36	156	348	343

CSV to temporal relational database



National Records of Scotland

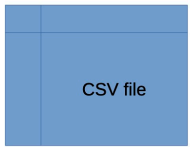
www.nrscotland.gov.uk/statistics-and-data

Links

Temporal database table

Week date	Sub-category	Fatality count	File_id	Date_valid_from	Date_valid_to

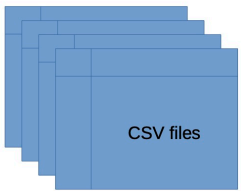
Asking different questions



?

How many deaths were reported in the week of 5 July?

How were those deaths distributed across the health boards?



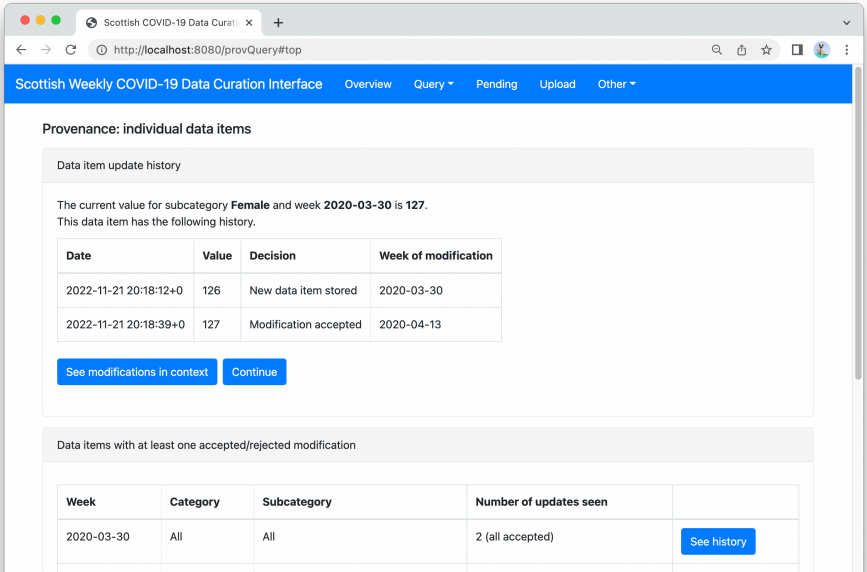
?

How many deaths were first reported in week of July 5?

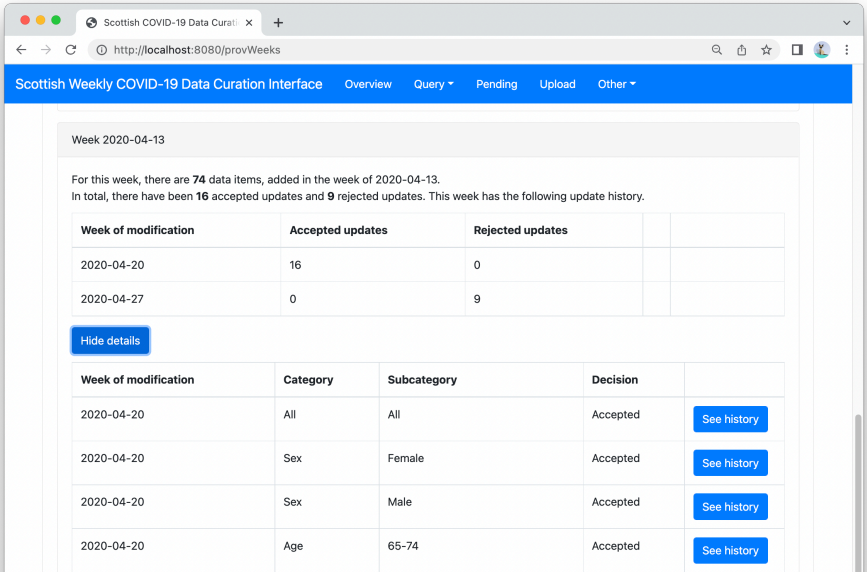
How does that differ from the most recently reported figure?

Which health boards reported the most changes?

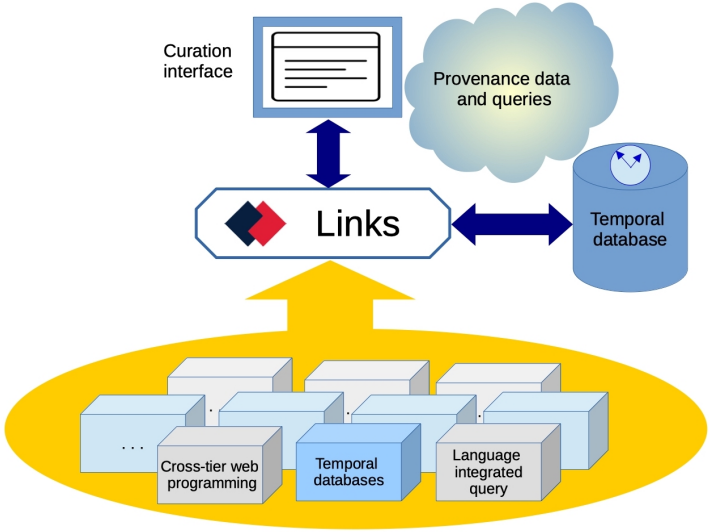
Links app screenshot: item provenance



Links app screenshot: week provenance



Using Links to build curation interfaces



Temporal Databases

Temporal features in SQL:2011

Krishna Kulkarni, Jan-Eike Michels (IBM Corporation)
{krishnak, janeike}@us.ibm.com

ABSTRACT

SQL:2011 was published in December of 2011, replacing SQL:2008 as the most recent revision of the SQL standard. This paper covers the most important new functionality that is part of SQL:2011: the ability to create and manipulate temporal tables.

1. Introduction

SQL is the predominant database query language standard published jointly by ISO (the International Organization for Standardization) and IEC (the International Electrotechnical Commission). In December 2011, ISO/IEC published the latest edition of the SQL standard, SQL:2011. A recent article in *SIGMOD Record* provides a brief survey of the new features in SQL:2011 [1]. Because of space constraints, it did not cover the most important new feature in SQL:2011: the ability to create and manipulate temporal tables, i.e., tables whose rows are associated with one or more temporal periods. This is the subject of the current article.

2. Temporal data support

Extensions to support temporal data[1] in SQL have long been desired. There is a large body of research papers, conference publications, and books on this topic, some dating back to the early 1980s. For more details, we refer the readers to an extensive (but outdated) bibliography [2] and to books such as [3] and [4].

Though the previous academic research produced a large number of solutions, the commercial adoption has been rather slow. It is only recently that commercial database management systems (DBMSs) have begun to offer SQL extensions for managing temporal data [5, 6, 7]. Prior to this development, users were forced to implement temporal support as part of the application logic, which often resulted in expensive development cycles and complex, hard-to-maintain code.

In 1995, the ISO SQL committee initiated a project to create a new part of SQL standard devoted to the language extensions for the temporal data support. A set of language extensions based on (but not identical to) TSQL2 [8] were submitted for standardization at that time. Unfortunately, these proposals generated considerable controversy (see [9] for more details), and failed to get adequate support from the ISO SQL committee's membership. In addition, there was no indication that key DBMS vendors were planning to implement these extensions in their products. Eventually, the work on this new part was cancelled in 2001.

Recently, a new set of language extensions for temporal data support were submitted to and accepted by the ISO SQL committee. These language extensions are now part of SQL:2011 Part 2, SQL/Foundation [10], instead of appearing as a new part. There is currently at least one commercial implementation [5] based on these extensions that the authors are aware of.

2.1 Periods

The cornerstone of temporal data support in SQL:2011 is the ability to define and associate time periods with the rows of a table. Essentially, a time period is a mathematical interval on the timeline, demarcated by a start time and an end time.

Many treatments of temporal databases introduce a period data type, defined as an ordered pair of two datetime values, for the purpose of associating time periods with the rows of a table. SQL:2011 has not taken this route. Adding a new data type to the SQL standard (or to an SQL product) is a costly venture because of the need to support the new data type in the tools and other software packages that form the ecosystem surrounding SQL. For example, if a period type were added to SQL, then it would have to also be added to the stored procedure language, to all database APIs such as JDBC, ODBC, and .NET, as well as to the surrounding technology such as ETL products, replication solutions, and others. There must also be some means of communicating period values to host languages that do not support period as a native data type, such as C or Java. These factors can potentially slow down the adoption of new type for a long time.

1. Note that there is no single, commonly accepted definition of the term "temporal data". For the purposes of this article, we define "temporal data" to mean any data with one or more associated time periods during which that data is deemed to be effective or valid along some time dimension.

SQL:2011 standardises SQL temporal extensions, but these are **not widely implemented**

Thus, users must implement temporal support themselves: brittle, error-prone

Developing Time-Oriented Database Applications in SQL

RICHARD T. SNODGRASS

Morgan Kaufmann Publishers
San Francisco, California

Dimensions of Time

User-defined time

→ Single field (for example, date of birth). Uninteresting for our purposes.

Transaction time

→ The time rows were present in the database.

Valid time

→ The time a row was valid in the domain being modelled.

(Bitemporal)

→ Transaction time and valid time used together.

Transaction-time: tracking database changes

Transaction time tables model when a fact was stored in a database.

→ Modification operations manipulate ‘recorded from’ and ‘recorded to’ fields.
→ Users do **not** modify the ‘recorded from’ and ‘recorded to’ fields directly.
→ Timestamps refer only to the **past**.
→ Database grows monotonically: data never lost.

☑ Go shopping
☐ Cook dinner
☐ Walk the dog
☐ Watch TV

task	done	from	to
Go shopping	t	11:00	∞
Cook dinner	f	11:00	∞
Walk the dog	f	11:00	∞
Watch TV	f	11:00	∞

Transaction Time Modifications

Check off “cook dinner”

- ☑ Go shopping
- ☑ Cook dinner
- ☐ Walk the dog
- ☐ Watch TV

```
update ( x <-t- todoTbl)
  where (ttData(x).task == "Cook dinner")
    set (done = true)
```

task	done	from	to
Go shopping	t	11:00	∞
Cook dinner	f	11:00	17:30
Walk the dog	f	11:00	∞
Watch TV	f	11:00	∞
Cook dinner	t	17:30	∞

Transaction Time Modifications

...then delete “Watch TV”

- ☑ Go shopping
- ☑ Cook dinner
- ☐ Walk the dog

```
delete (x <-t- todoTbl)
  where (ttData(x).task == "Watch TV")
```

task	done	from	to
Go shopping	t	11:00	∞
Cook dinner	f	11:00	17:30
Walk the dog	f	11:00	∞
Watch TV	f	11:00	19:00
Cook dinner	t	17:30	∞

Transaction Time Queries

What was the state of the to-do list at 18:00?

```
query { for (x <-t- todoTbl)
  where ((time >= ttFrom(x)) && (time < ttTo(x)))
    [ttData(x)]};
```

or equivalently

```
query nested { for (x <- ttAt(todoTbl,time)) [x]}
```

task	done
Go shopping	t
Cook dinner	t
Walk the dog	f
Go shopping	f

Transaction Time Queries

When was it recorded that “Cook dinner” was completed?

```
query { for (x <-t- todoTbl)
  where ((ttData(x).task=="Cook dinner")
     &&  (ttData(x).done==true))
    [(time=ttFrom(x))]};
```

time
17:30

Transaction Time Queries

Give the change history for task “Cook dinner”.

```
query { for (x <-t- todoTbl)
  where (ttData(x).task=="Cook dinner")
    [(task=ttData(x).task,done=ttData(x).done,
      tt_from=ttFrom(x),tt_to=ttTo(x))]};
```

task	done	tt_from	tt_to
Cook dinner	f	11:00	17:30
Cook dinner	t	17:30	∞

Valid Time Tables

Valid-time: tracking real-world changes

Valid time tables model when a fact was true in the modelled reality.

- → Timestamps can refer to **future** as well as past
- → Users **can** manipulate 'valid from' and 'valid to' fields directly (for example, extending an employment contract)
- → Ability to apply an update or deletion over a time period: **sequenced** updates

Name	Position	Salary	from	to
Alice	Lecturer	£40000	2000	2008
Alice	Senior Lecturer	£50000	2008	∞
Bob	PhD Student	£15000	2019	2023

Valid Time Modifications

Name	Position	Salary	from	to
Alice	Lecturer	£40000	2000	2008
Alice	Senior Lecturer	£50000	2008	∞
Bob	PhD Student	£15000	2019	2023
Carol	Professor	£70000	2022	∞

Hire Carol as a professor, on a permanent contract.
(**Current insert**)

insert employees **values** (Name $=$ **"Carol"**, Position $=$ **"Professor"**, Salary $= 70000$)

Valid Time Modifications

Name	Position	Salary	from	to
Alice	Lecturer	£40000	2000	2008
Alice	Senior Lecturer	£50000	2008	2022
Bob	PhD Student	£15000	2019	2023
Carol	Professor	£70000	2022	∞

Alice has resigned.
(**Current deletion**)

delete (x $\Leftarrow$ employees) **where** x.name $==$ "Alice"

Valid Time Modifications

Name	Position	Salary	from	to
Alice	Lecturer	£40000	2000	2008
Alice	Senior Lecturer	£50000	2008	2022
Bob	PhD Student	£15000	2019	2023
Carol	Professor	£70000	2022	2023
Carol	Head of School	£70000	2023	2026
Carol	Professor	£70000	2026	∞

Carol will be Head of School between 2023 and 2026.
(**Sequenced Update**)

update (x $\Leftarrow$ employees)
  between $(2023, 2026)$
  where (x.name $==$ "Carol")
  set (position $=$ "Head of School")

Valid Time Modifications

Name	Position	Salary	from	to
Alice	Lecturer	£40000	2000	2008
Alice	Senior Lecturer	£50000	2008	2020
Bob	PhD Student	£15000	2019	2024
Carol	Professor	£70000	2022	2023
Carol	Head of School	£70000	2023	2026
Carol	Professor	£70000	2026	∞

Extend Bob's contract until 2024.
(**Nonsequenced Update**)

update nonsequenced (x $\leftarrow$ employees)
  where ((**data** x).name $==$ "Bob")
  set (x **valid to** 2024)

Formalism

Key idea

Define direct semantics for temporal operations, and show translations into non-temporal DB-supporting λ-calculus

$$\mathsf{M} \Downarrow^{\mathsf{T}}_{\Delta,\iota} (\mathsf{V}, \Delta')$$

Evaluate term M wrt. database Δ at timestamp ι, resulting in value V and updated database Δ'

Formalism Example (λ_{TLINQ} update)

$$\frac{\mathsf{L} \Downarrow^{\mathsf{T}}_{\Delta,\iota} (\mathsf{t}, \Delta_1) \qquad \Delta_2 = \Delta_1[\mathsf{t} \mapsto \hat{\bigoplus} \wr\, \mathsf{upd}(\mathsf{v}) \mid \mathsf{v} \in \Delta_1(\mathsf{t}) \int]}{\begin{array}{l}\mathsf{upd}(\textbf{data}^{[\textbf{start},\textbf{end})}) = \\ \quad \begin{cases} \wr \mathsf{data}^{[\mathsf{start},\iota)}, (\textbf{data with } \overrightarrow{\ell = \mathsf{V}})^{[\iota,\infty)} \int \\ \quad \text{if } \mathsf{M}\{\mathsf{data}/\mathsf{x}\} \Downarrow^{\star}_{\iota} \textbf{true}, (\mathsf{N_i}\{\mathsf{data}/\mathsf{x}\} \Downarrow^{\star}_{\iota} \mathsf{V_i})_\mathsf{i}, \text{and } \textbf{end} = \infty \\ \wr \textbf{data}^{[\textbf{start},\textbf{end})} \int \quad \text{otherwise} \end{cases}\end{array}}{\textbf{update } (\mathsf{x} \Leftarrow \mathsf{L}) \textbf{ where } \mathsf{M} \textbf{ set } (\overrightarrow{\ell = \mathsf{N}}) \Downarrow^{\mathsf{T}}_{\Delta,\iota} ((), \Delta_2)}$$

For every record where the predicate matches, close off existing row and add updated row (valid from now until forever)

Formalism Example (Translation)

$$
\begin{array}{l}
[\![\textbf{update}^{(\ell_i:A_i)_{i\in I}}\ (x \Leftarrow L)\ \textbf{where}\ M\ \textbf{set}\ (\ell = N_j)_{j\in J}]\!] = \\
\quad \textbf{let}\ tbl = [\![L]\!]\ \textbf{in} \\
\quad \textbf{let}\ affected = \textbf{query} \\
\qquad \textbf{for}\ (x \leftarrow \textbf{get}\ tbl) \\
\qquad\quad \text{where}\ ((\text{restrict}(x, \{\ell_i\}_{i\in I}, [\![M]\!]) \wedge \text{isCurrent}(x))) \\
\qquad\quad \left(\begin{array}{l}(\ell_i = x.\ell_i)_{i\in I/J} \oplus \\ (\ell_j = \text{restrict}(x, \{\ell_i\}_{i\in I}, [\![N_j]\!]))_{j\in J} \oplus \\ (\textbf{start} = \textbf{now}, \textbf{end} = \infty)\end{array}\right) \\
\quad \textbf{in} \\
\quad \textbf{update}\ (x \Leftarrow tbl) \\
\qquad \textbf{where}\ (\text{restrict}(x, \overrightarrow{\ell}, [\![M]\!]) \wedge \text{isCurrent}(x)) \\
\qquad \textbf{set}\ (\textbf{end} = \textbf{now}); \\
\quad \textbf{insert}\ tbl\ \textbf{values}\ affected
\end{array}
$$

- → Calculate set of affected rows (**query**)
- → Close off existing rows (**update**)
- → Insert affected rows (**insert**)

- → (restrict used to η-expand translated record to ensure type correctness)

Metatheory (λ_{TLINQ})

Preservation

→ A well-typed closed term evaluated wrt. a well-formed database will always evaluate to a value paired with another well-formed database

Semantics Preservation

→ The translation of a well-typed, closed λ_{TLINQ} term and well-formed database will evaluate to a value and database corresponding to the result of evaluating using the native semantics

Similar results hold for λ_{VLINQ}

Sequenced Joins

Joining with a nontemporal table

name	band	from	to
Alice	A08	2000	2008
Alice	A09	2008	2020
Bob	B01	2019	2024

band	salary
B01	£15000
A07	£33000
A08	£40000
A09	£50000
UE10	£70000

Pair names of employees with their salaries

name	salary	from	to
Alice	£40000	2000	2008
Alice	£50000	2008	2020
Bob	£15000	2019	2024

Joining with a temporal table

name	band	from	to
Alice	A08	2000	2008
Alice	A09	2008	2020
Bob	B01	2019	2024

band	salary	from	to
B01	£15000	2000	2015
A07	£33000	2000	2015
A08	£40000	2000	2015
A09	£50000	2000	2015
B01	£16000	2015	∞
A07	£34000	2015	∞
A08	£41000	2015	∞
A09	£51000	2015	∞

Pair names of employees with their salaries at each point in time

Joining with a temporal table

name	band	from	to
Alice	A08	2000	2008
Alice	A09	2008	2020
Bob	B01	2019	2024

band	salary	from	to
B01	£15000	2000	2015
A07	£33000	2000	2015
A08	£40000	2000	2015
A09	£50000	2000	2015
B01	£16000	2015	∞
A07	£34000	2015	∞
A08	£41000	2015	∞
A09	£51000	2015	∞

Pair names of employees with their salaries at each point in time

name	salary	from	to
Alice	£40000	2000	2008
Alice	£50000	2008	2015
Alice	£51000	2015	2020
Bob	£16000	2019	2024

Joining with a temporal table

name	band	from	to
Alice	A08	2000	2008
Alice	A09	2008	2020
Bob	B01	2019	2024

band	salary	from	to
B01	£15000	2000	2015
A07	£33000	2000	2015
A08	£40000	2000	2015
A09	£50000	2000	2015
B01	£16000	2015	∞
A07	£34000	2015	∞
A08	£41000	2015	∞
A09	£51000	2015	∞

Pair names of employees with their salaries at each point in time

- → Straightforward to encode for flat queries using LINQ
- → Simple rewrite using greatest and least functions to calculate the intersections of the periods of validity

Temporal case study: XML documents

Temporal case study: XML documents

A different context

- → NPL (National Physical Laboratory) is Britain's National Metrology Institute
- → Measuring devices require calibration certificates that provide traceability back to SI units
- → The digital transformation of metrology is ongoing: recent XML schema for digital calibration certificates

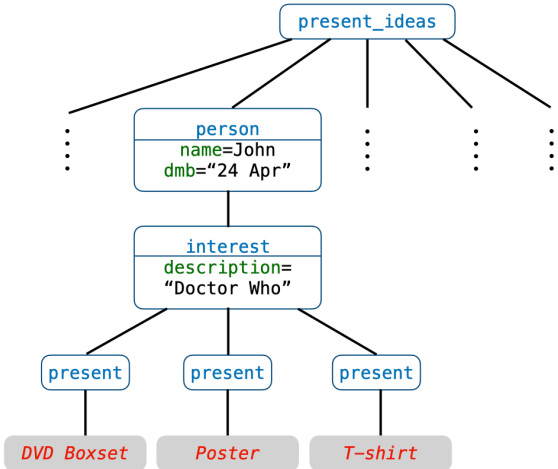
Research project

- → How can Links with temporal features be applied in this domain?
- → Prototype curation interface developed for XML documents
- → Well-received at metrology conference last year
- → Recent funding to develop this further

Demo

XML: Extensible Markup Language

```
<present_ideas>
  <person name="John" dmb="24 Apr”/>
  <person name="Fred" dmb="03 Jun">
    <interest description = "Doctor Who">
      <present>DVD Boxset</present>
      <present>Poster</present>
      <present>T-shirt</present>
    </interest>
  </person>
  <person name="Amina" dmb="04 Jun”/>
  <person name="Kim" dmb="10 Sep">
      <present>Voucher</present>
  <person name="May" dmb="25 Dec”/>
</present_ideas>
```



Implementation choices

Links: relational database

→ 1. Use XML schema to determine database tables
→ 2. Use a schema-agnostic approach to store XML ✓

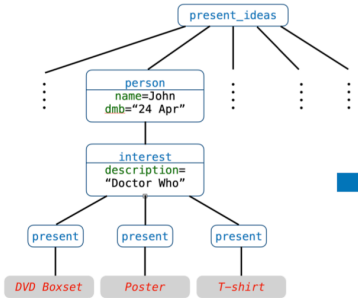
Dynamic Dewey labelling [Xu et al (SIGMOD 2009)]

→ Comparison of labels to determine relationship between nodes
→ Unique label for each node and efficient for querying
→ No relabelling needed for insertion so more efficient than others
→ Straightforward to implement

Evaluation

→ User interface could be useful for people who don't know XML
→ Schema-agnosticism was the right choice because the schema is not final

XML labelling for relational storage



xml_node

Label	Tag	Text	...
+001	present_ideas		
+001.+002	person		
+001.+002.+003	interest		
+001.+002.+003.+002	present		
+001.+002.+003.+002.+001		DVD Boxset	

xml_attr

Label	Attr	Value	...
+001.+002	name	John	
+001.+002	dmb	24 Apr	
+001.+002.+003	description	Doctor Who	

Demo

Wrapping up

How to install Links

Using opam 4.14.1

```
> opam switch create 4.14.1
> eval $(opam env --switch=4.14.1); opam switch 4.14.1
> opam install links
```

Install database driver: `links-postgresql` or `links-sqlite3`
Database instructions for postgreSQL:
https://github.com/links-lang/links/wiki/Database-setup

Database and opam installation for macOS

Install HomeBrew: https://brew.sh

```
> brew install opam
```

Install database with brew: `postgresql` or `sqlite3`

These instructions were correct as of 24 July 2023 but may now be out-of-date. See https://github.com/links-lang for current instructions.

Resources

Software

→ The Links Programming Language: https://links-lang.org

Case Studies

→ Covid 19 data: https://doi.org/10.5281/zenodo.7199221
→ XML documents: https://doi.org/10.5281/zenodo.7551691

Tutorial material

→ Slides and tutorial code:
https://github.com/links-lang/splv2023_example_code

References I

- Peter Buneman, Shamim A. Naqvi, Val Tannen, and Limsoon Wong. Principles of programming with complex objects and collection types, TCS 149, 1995. https://doi.org/10.1016/0304-3975(95)00024-Q
- James Cheney and Wilmer Ricciotti, Comprehending Nulls, DBPL 2021. https://doi.org/10.1145/3475726.3475730
- James Cheney, Sam Lindley, Gabriel Radanne, and Philip Wadler, Effective quotation: relating approaches to language-integrated query, PEPM 2014. https://doi.org/10.1145/2543728.2543738
- James Cheney, Sam Lindley, and Philip Wadler, Query shredding: efficient relational evaluation of queries over nested multisets, SIGMOD 2014. https://doi.org/10.1145/2588555.2612186
- James Cheney, Sam Lindley, and Philip Wadler. A practical theory of language-integrated query, ICFP 2013. https://doi.org/10.1145/2500365.2500586
- Kwanghoon Choi, James Cheney, Sam Lindley, and Bob Reynders, A Typed Slicing Compilation of the Polymorphic RPC Calculus, PPDP 2021. https://doi.org/10.1145/3479394.3479406
- Kwanghoon Choi, James Cheney, Simon Fowler, and Sam Lindley, A Polymorphic RPC Calculus, SCP 197, 2020. https://doi.org/10.1016/j.scico.2020.102499
- Ezra Cooper, The Script-Writer's Dream: How to Write Great SQL in Your Own Language, and Be Sure It Will Succeed, DBPL 2009. https://doi.org/10.1007/978-3-642-03793-1_3
- Ezra Cooper and Philip Wadler, The RPC calculus, PPDP 2009. https://doi.org/10.1145/1599410.1599439
- Ezra Cooper, Sam Lindley, Philip Wadler and Jeremy Yallop, The essence of form abstraction. APLAS 2008. https://doi.org/10.1007/978-3-540-89330-1_15

References II

- → Ezra Cooper, Sam Lindley, Philip Wadler, and Jeremy Yallop, Links: web programming without tiers, FMCO 2006. https://doi.org/10.1007/978-3-540-74792-5_12
- → Simon Fowler, Vashti Galpin, and James Cheney, Language-Integrated Query for Temporal Data, GPCE 2022. https://doi.org/10.48550/arXiv.2210.12077
- → Simon Fowler, Simon Harding, Joanna Sharman, and James Cheney, Cross-tier web programming for curated databases: a case study, IJDC 2020. https://doi.org/10.2218/ijdc.v15i1.717
- → Simon Fowler, Model-View-Update-Communicate: Session Types meet the Elm Architecture, ECOOP 2020. https://doi.org/10.4230/LIPIcs.ECOOP.2020.14
- → Simon Fowler, Sam Lindley, J. Garrett Morris, and Sára Decova, Exceptional Asynchronous Session Types: Session Types without Tiers, POPL 2019. https://doi.org/10.1145/3290341
- → Stefan Fehrenbach and James Cheney, Language-integrated provenance by trace analysis, DBPL 2019. https://doi.org/10.1145/3315507.3330198
- → Stefan Fehrenbach and James Cheney, Language-integrated provenance, SCP 155 (2018). https://doi.org/10.1016/j.scico.2017.08.009
- → Stefan Fehrenbach and James Cheney, Language-integrated provenance, PPDP 2016. https://doi.org/10.1145/2967973.2968604
- → Vashti Galpin, Ian Smith, Jean-Laurent Hippolyte, Tracking and viewing modifications in digital calibration certificates, Acta IMEKO, 2023. https://doi.org/10.21014/actaimeko.v12i1.1407
- → Vashti Galpin and James Cheney, Curating Covid-19 data in Links, IPAW 2021 demo paper. https://doi.org/10.1007/978-3-030-80960-7_19

References III

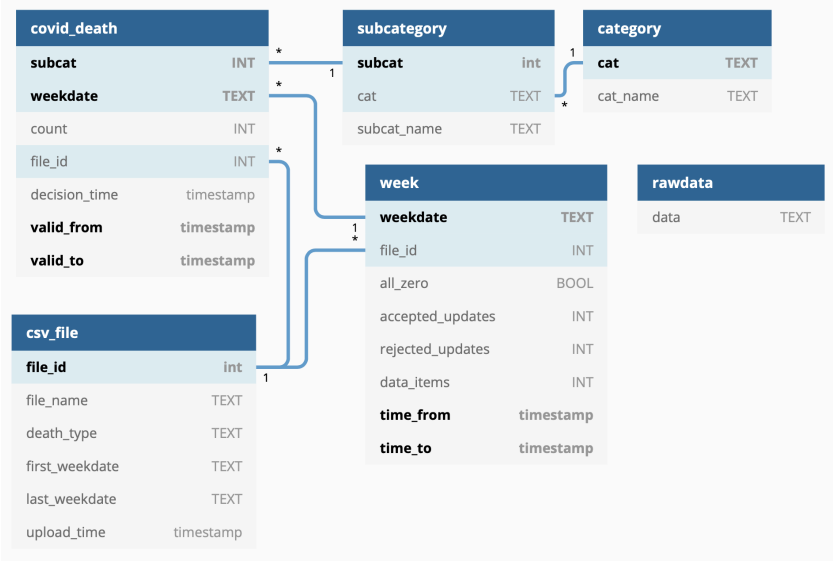
- Daniel Hillerström, Sam Lindley, and John Longley, Effects for Efficiency: Asymptotic Speedup with First-Class Control, ICFP 2020. https://doi.org/10.1145/3408982
- Daniel Hillerström, Sam Lindley, and Robert Atkey, Effect handlers via generalised continuations, JFP 30, 2020. https://doi.org/10.1017/S0956796820000040
- Daniel Hillerström and Sam Lindley, Shallow Effect Handlers, APLAS 2018. https://doi.org/10.1007/978-3-030-02768-1_22
- Daniel Hillerström, Sam Lindley, Robert Atkey, and KC Sivaramakrishnan, Continuation Passing Style for Effect Handlers, FSCD 2017. https://doi.org/10.4230/LIPIcs.FSCD.2017.18
- Daniel Hillerström and Sam Lindley, Liberating Effects with Rows and Handlers, TyDe 2016. https://doi.org/10.1145/2976022.2976033
- Rudi Horn, Simon Fowler and James Cheney, Language-Integrated Updatable Views, IFL 2019. https://doi.org/10.1145/3236769
- Rudi Horn, Roly Perera, and James Cheney, Incremental Relational Lenses, ICFP 2018. https://doi.org/10.1145/3412932.3412945
- Sam Lindley and J. Garrett Morris, Lightweight functional session types, Behavioural Types: from Theory to Tools, 2017.
- Sam Lindley and James Cheney, Row-based effect types for database integration, TLDI 2012. https://doi.org/10.1145/2103786.2103798
- Erik Meijer, Brian Beckman and Gavin M. Bierman. LINQ: reconciling object, relations and XML in the .NET framework, SIGMOD 2006. https://doi.org/10.1145/1142473.1142552

References IV

- → Richard T. Snodgrass, Developing Time-Oriented Database Applications in SQL, 2000, Morgan-Kaufman
- → Don Syme, Leveraging .NET meta-programming components from F#: integrated queries and interoperable heterogeneous execution, ML Workshop, 2006. https://doi.org/10.1145/1159876.1159884
- → Wilmer Ricciotti and James Cheney, A Formalization of SQL with Nulls, JAR 66, 2022. https://doi.org/10.1007/s10817-022-09632-4
- → Wilmer Ricciotti and James Cheney, Strongly-Normalizing Higher-Order Relational Queries. LMCS 18, 2022. https://doi.org/10.46298/lmcs-18(3:23)2022
- → Wilmer Ricciotti and James Cheney, Query Lifting: Language-integrated query for heterogeneous nested collections, ESOP 2021. https://doi.org/10.1007/978-3-030-72019-3_21
- → Wilmer Ricciotti and James Cheney, Strongly Normalizing Higher-Order Relational Queries, FSCD 2020. https://doi.org/10.4230/LIPIcs.FSCD.2020.28
- → Wilmer Ricciotti and James Cheney, Mixing Set and Bag Semantics, DBPL 2019. https://doi.org/10.1145/3315507.3330202
- → Limsoon Wong, Kleisli, a functional query system. JFP 10, 2000. https://doi.org/10.1017/s0956796899003585
- → Liang Xu, Tok Wang Ling, Huayu Wu and Zhifeng Bao, DDE: from Dewey to a fully dynamic XML labeling scheme, SIGMOD 2009 https://doi.org/10.1145/1559845.1559921

Details of Covid case study

Database schema



Example queries

```
# Sequenced flat query

   var subcat_of_interest = 8;                          # subcategory: age 65-74
   var week_of_interest = "2020-04-13";

                                                        # for any row from the
   for (y <-v- covid_death)                             # 'covid_death' table
      where
        (vtData(y).subcat==subcat_of_interest           # where the subcategory
      && vtData(y).weekdate==week_of_interest)          # and the week match

         [(subcat=vtData(y).subcat,                     # create a record with the
           weekdate=vtData(y).weekdate,                 # subcategory, week date,
           count=vtData(y).count,                       # death count, file_id and
           file_id=vtData(y).file_id,                   # validity period
           valid_from=vtFrom(y),
           valid_to=vtTo(y)
         )]
```

```
[(subcat = 8, weekdate = "2020-04-13", count = 80, file_id = 25,
    valid_from = 2021-07-21 11:36:25.955976,
      valid_to = 2021-07-21 12:26:47.082969),
 (subcat = 8, weekdate = "2020-04-13", count = 81, file_id = 26,
    valid_from = -infinity,
      valid_to = -infinity),
 (subcat = 8, weekdate = "2020-04-13", count = 81, file_id = 27,
    valid_from = 2021-07-21 12:26:47.082969,
      valid_to = infinity),
]
: [(subcat:Int,weekdate:String,count:Int,file_id:Int,
      valid_from:DateTime,
        valid_to:DateTime)]
```

Example queries

```
# Current time-slice nested query

   for (x <-- subcategory)                          # for any row in the 'subcategory'
     where (x.cat=="Sex")                           # table that has category "Sex"

       [(subcat_name=x.subcat_name,                 # create a record with the
         cat=x.cat,                                 # subcategory name and its category
                                                    # together with a list of the
         results=                                   # current values for each week
          for (y <-v- covid_death)                  # from the 'covid_death' table

             where (vtData(y).subcat==x.subcat      # where the subcategory matches
                 && vtTo(y)==forever())             # and the week data is valid

              [(count=vtData(y).count,              # create a record with the
                weekdate=vtData(y).weekdate)]       # death count and the week date
        )]
```

```
[(subcat_name = "Female",cat = "Sex",
results = [(count = 5, weekdate = "2020-03-16"), (count = 26, weekdate = "2020-03-23"), (count =
127, weekdate = "2020-03-30"), (count = 262, weekdate = "2020-04-06"), (count = 310, weekdate =
"2020-04-13"), (count = 346, weekdate = "2020-04-20"), (count = 279, weekdate = "2020-04-27"),
(count = 222, weekdate = "2020-05-04")]
),
(subcat_name = "Male",cat = "Sex",
results = [(count = 5, weekdate = "2020-03-16"), (count = 36, weekdate = "2020-03-23"), (count =
155, weekdate = "2020-03-30"), (count = 348, weekdate = "2020-04-06"), (count = 340, weekdate =
"2020-04-13"), (count = 313, weekdate = "2020-04-20"), (count = 247, weekdate = "2020-04-27"),
(count = 193, weekdate = "2020-05-04")]
)]
: [(subcat_name:String,cat:String,results:[(count:Int,weekdate:String)],)]
```

Example queries

```
# Sequenced inserts, deletes and updates for valid time


vt_insert covid_death values                                  # insert a new record
  (subcat,weekdate,count,file_id,decision_time)               # with valid time from the
    [withValidity(                                            # current time until forever
      ( <values to be inserted> ),
      now(),forever())]

for (x <- decs)                                               # from a list of decisions
  update sequenced (y <-v- covid_death)                       # update a count for specific
    between (now(),forever())                                 # subcategory and week
      where (x.weekdate==y.weekdate && x.subcat==y.subcat)     # while retaining the
        set (count = x.new_value, file_id=x.file_id)].        # previous count

#-------------------------------------------------------------------------------


update nonsequenced (x <-v- covid_death)                      # update the count to zero
   where (x.cat=="Location")                                  # for all records with the
   set (count = 0)                                            # category "Location"

delete sequenced (x <-v- covid_death)                         # delete records in
   between (error_from, error_to)                             # a specific category
   where (x.cat == "Location")                                # during a given time period

delete nonsequenced (x <-v- covid_death) where (true)         # delete all records

# these three queries destroy update provenance information!
```