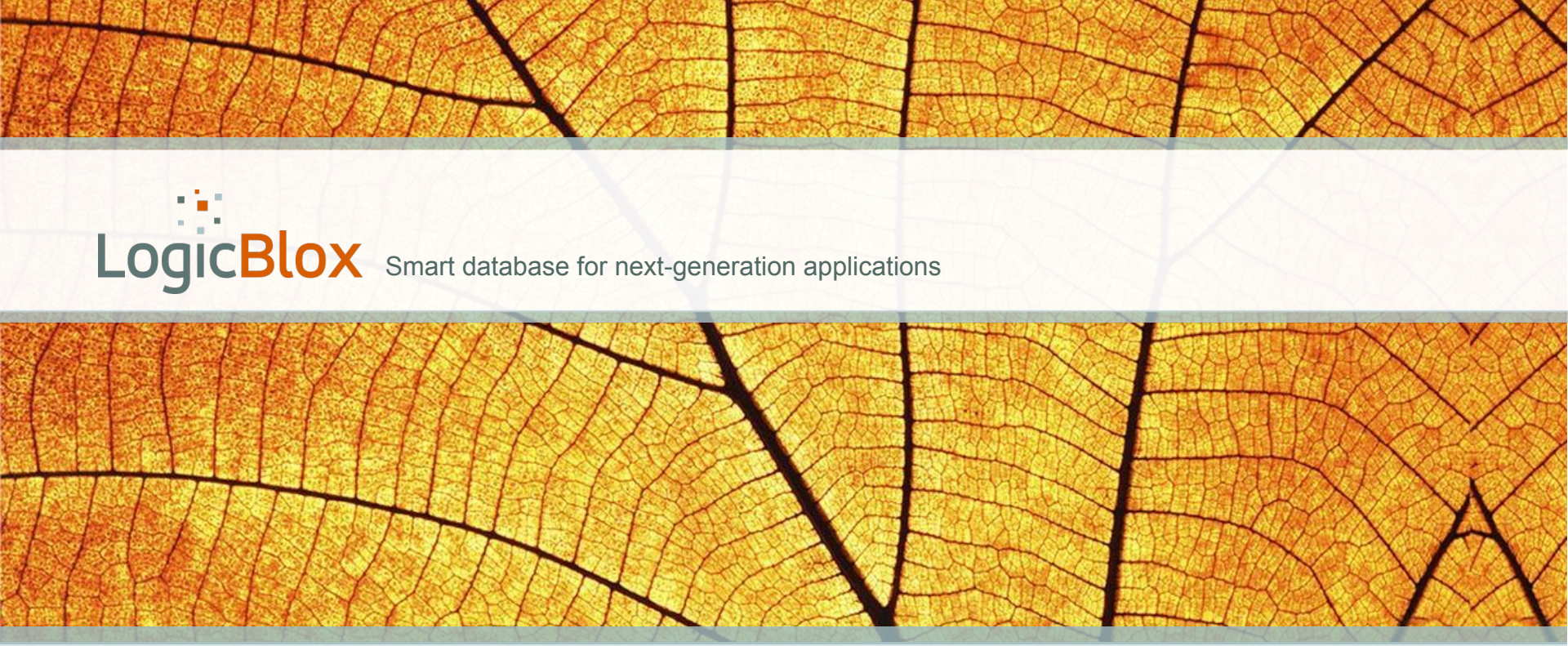


LogicBlox Smart database for next-generation applications

LESSON 6 : AGGREGATIONS AND HIERARCHIES

LOGICBLOX TRAINING

SUMMARY SO FAR (1)

- model
  - predicates: fundamental data structure
    - IDB (internal state) and EDB (external state)
    - primitive types & entities (user-defined types)
  - constraints

- logic
  - IDB rules
  - EDB (delta) rules: event handling, data import

SUMMARY SO FAR (2)

- code organization
  - project
  - modules
- interface with LogicBlox
  - lb: command line
  - lb web service container
- testing
  - lb unit (logic)
  - lb.web.testcase.* (services)

NEXT STEPS : DEEPER DIVES

- lesson 6 : more computational power
  - aggregations, hierarchies
- lesson 7 : scalable data import/export
  - delimited file service

AGGREGATIONS IN LOGICBLOX

- count
- total
- min
- max
- top/n
- bottom/n
- concat (for strings)
- and
- or
- ambig

AGGREGATIONS IN LOGICBLOX

- count
- total
- min
- max
- top/n
- bottom/n
- concat (for strings)
- and
- or
- ambig

COUNT

- total number of wines

```
numWines[] = n
  <- agg<< n = count() >> wine(_).
```

- SQL analogy

```
SELECT count(*)
FROM wine
```

COUNT, WITH GROUP BY

- total number of wines, by vintage year

```
numWines_byYear[year] = n
  <- agg<< n = count() >>
     wine:ofYear(wine,year).
```

- SQL analogy

```
SELECT year, count(wine)
FROM wine:ofYear.
```

IN-CLASS EXERCISE 1

- count wine by type

- cheat sheet

```
numWines_byYear[year] = n
  <- agg<< n = count() >>
     wine:ofYear(wine,year).
```

IN-CLASS EXERCISE 1 ANSWER

- count wine by type

```
numWines_byType[type] = n
  <- agg<< n = count() >>
     wine:ofType(wine,type).
```

```
% lb print discountwines core:wine:numWines_byType
[2] "red"     2
[3] "white"   2
```

IN-CLASS EXERCISE 2

- count wine by vintage year and type

- cheat sheet

```
numWines_byYear[year] = n
  <- agg<< n = count() >>
     wine:ofYear(wine,year).
```

IN-CLASS EXERCISE 2 ANSWER

- count wine by vintage year and type

```
numWines_byYearType[year,type] = n
  <- agg<< n = count() >>
     wine:ofYear(wine,year),
     wine:ofType(wine,type).

$ lb print discountwines core:wine:numWines_byYearType
[0] 2007   [2] "red"     1
[3] 2008   [2] "red"     1
[1] 2010   [3] "white"   2
```

TOTAL AGGREGATIONS

- first we need something to total

logiql/core/sales.logic

```
block(`sales) {
  export(`{
    unit_sales[wine, day] = units
      -> wine(wine),
         hierarchies:calendar:day(day),
         int(units).
  }
} <-- .
```

TOTAL AGGREGATIONS

- total all by sales

```
unit_sales_total[] = tunits
  <- agg<< tunits = total(units) >>
     unit_sales[_, _] = units.
```

- SQL analogy

```
SELECT sum(units)
FROM unit_sales
```

IN-CLASS EXERCISE 3

- total unit_sales group by day

- cheat sheet
  - total unit sales

```
total_sales[] = tunits
  <- agg<< tunits = total(units) >>
     unit_sales[wine, day] = units.
```

IN-CLASS EXERCISE 3 ANSWERS

- total unit_sales group by day

```
unit_sales_day[day] = units
  <- agg<< tunit = total(units) >>
     unit_sales[_, day] = units.
```

- cheat sheet
  - total unit sales

```
total_sales[] = tunits
  <- agg<< tunits = total(units) >>
     unit_sales[wine, day] = units.
```

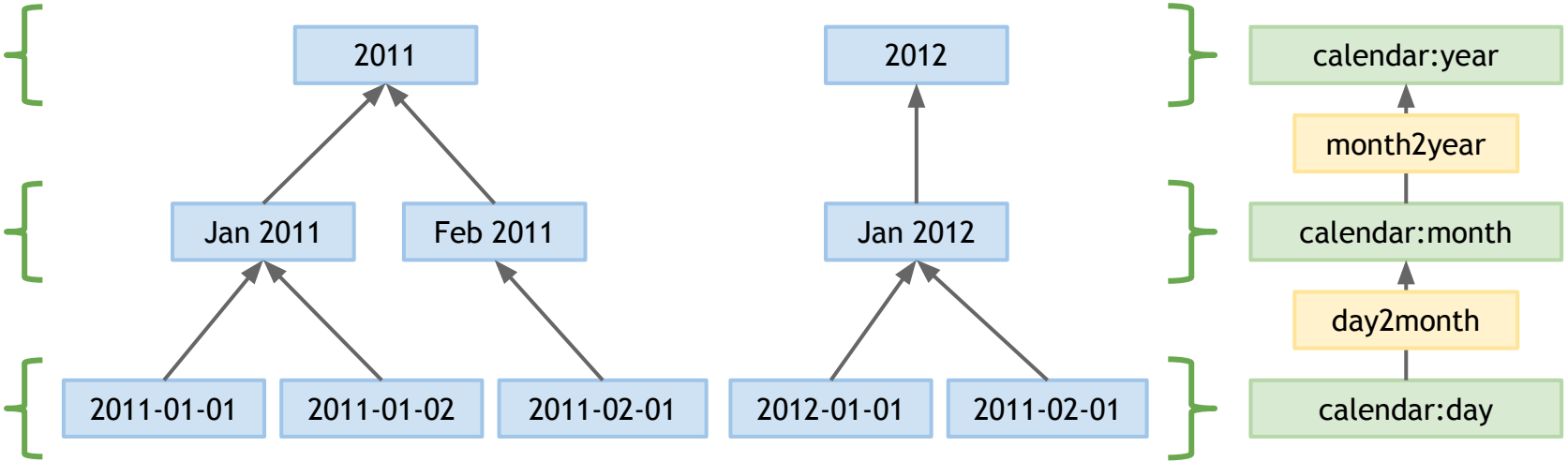
A BIT MORE CALENDAR STRUCTURE

logiql/core/hierarchies/calendar.logic

```
block(`calendar) {
  export(`{
    day(x), day_id(x : id) -> string(id).
    month(x), month_id(x : id) -> string(id).
    year(x), year_id(x : id) -> string(id).

    day2month[x] = y -> day(x), month(y).
    month2year[x] = y -> month(x), year(y).
   }
} <-- .
```

CALENDAR RELATIONSHIPS



AGGREGATION USING CALENDAR

- total unit_sales group by wine, month

```
unit_sales_wine_month[wine, month] = munits
  <- agg<< munits = total(units) >>
     unit_sales[wine, day] = units,
     day2month[day] = month.
```

- total unit_sales group by wine, year

```
unit_sales_wine_year[wine, year] = munits
  <- agg<< munits = total(units) >>
     unit_sales[wine, day] = units,
     day2month[day] = month,
     month2year[month] = year.
```

IN-CLASS EXERCISE 4

- total unit_sales group by wine, year

```
unit_sales_wine_year[wine, year] = munits
  <- agg<< munits = total(units) >>
     unit_sales[wine, day] = units,
     day2month[day] = month,
     month2year[month] = year.
```

- what's an alternative way of defining this?

IN-CLASS EXERCISE 4 ANSWER

- total unit_sales group by wine, year

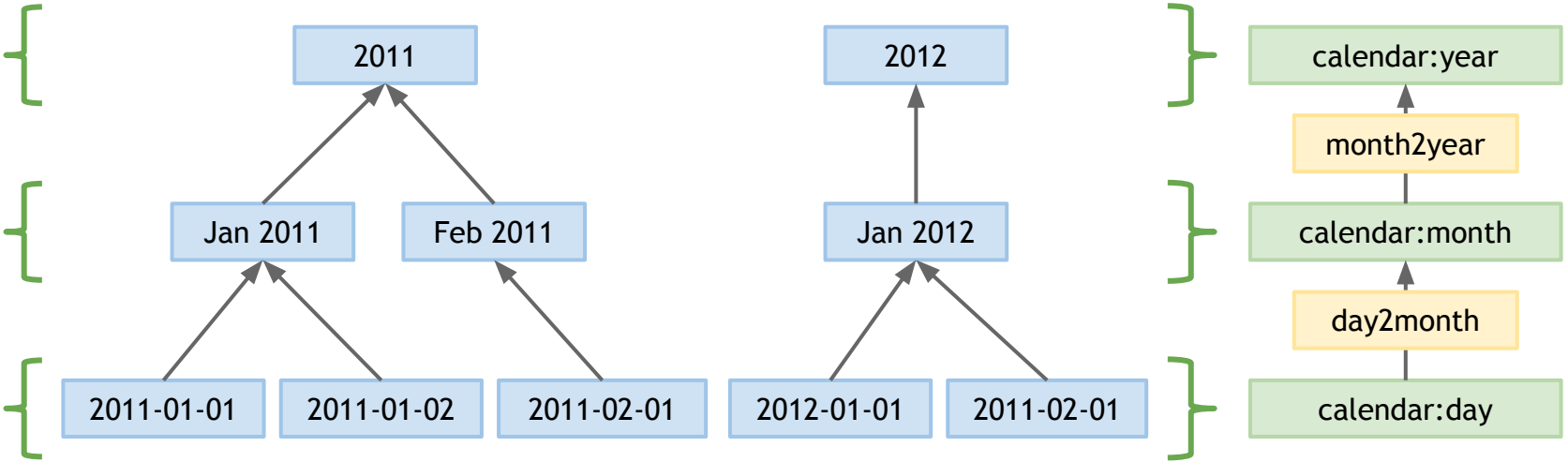
```
unit_sales_wine_year[wine, year] = munits
  <- agg<< munits = total(units) >>
     unit_sales[wine, day] = units,
     day2month[day] = month,
     month2year[month] = year.
```

- what's an alternative way of defining this?

```
unit_sales_wine_year[wine, year] = munits
  <- agg<< munits = total(units) >>
     unit_sales_wine_month[wine, month] = units,
     month2year[month] = year.
```

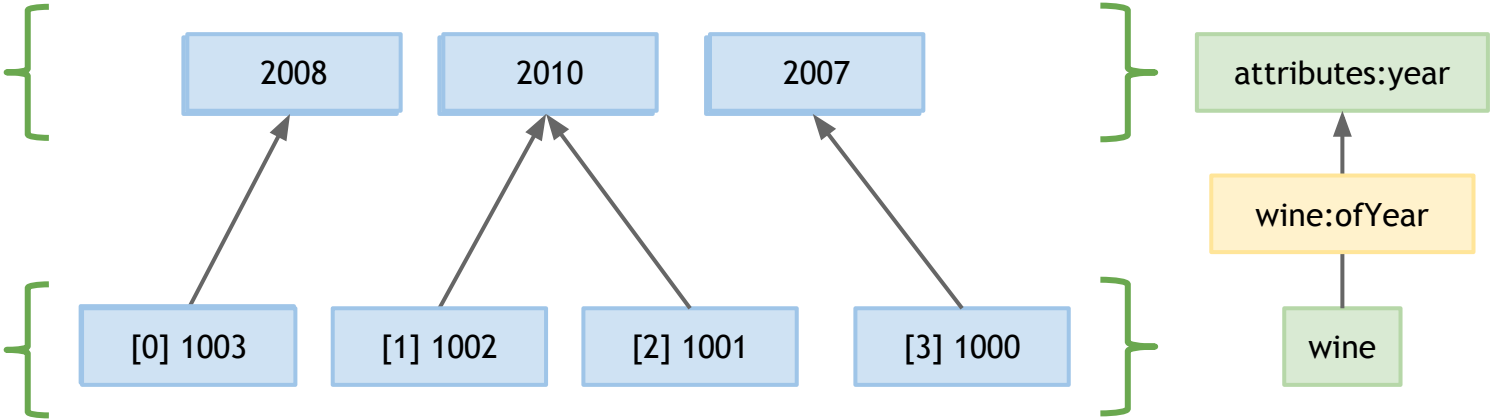
FOUNDATIONS FOR MEASURES

```
unit_sales[wine, day] = units
  -> wine(wine),
     hierarchies:calendar:day(day),
     int(units).
```



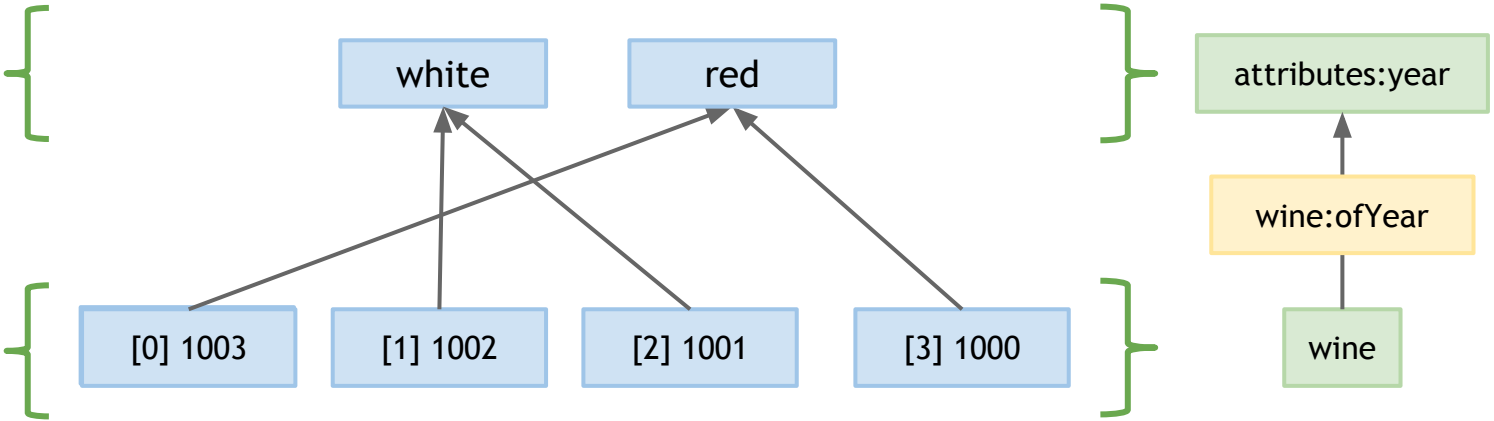
FOUNDATIONS FOR MEASURES

```
unit_sales[wine, day] = units
  -> wine(wine),
     hierarchies:calendar:day(day),
     int(units).
```



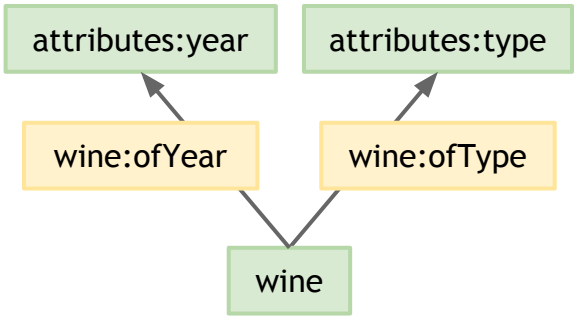
FOUNDATIONS FOR MEASURES

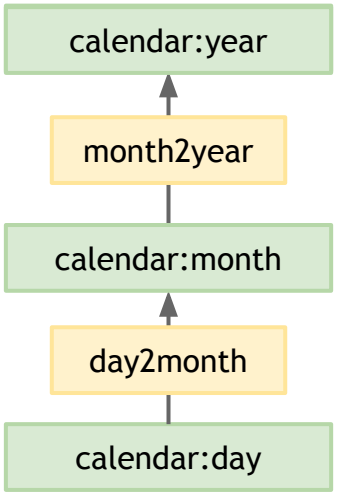
```
unit_sales[wine, day] = units
  -> wine(wine),
     hierarchies:calendar:day(day),
     int(units).
```



FOUNDATIONS FOR MEASURES

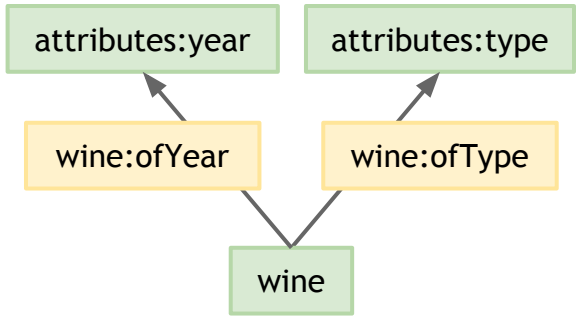
```
unit_sales[wine, day] = units
  -> wine(wine),
     hierarchies:calendar:day(day),
     int(units).
```

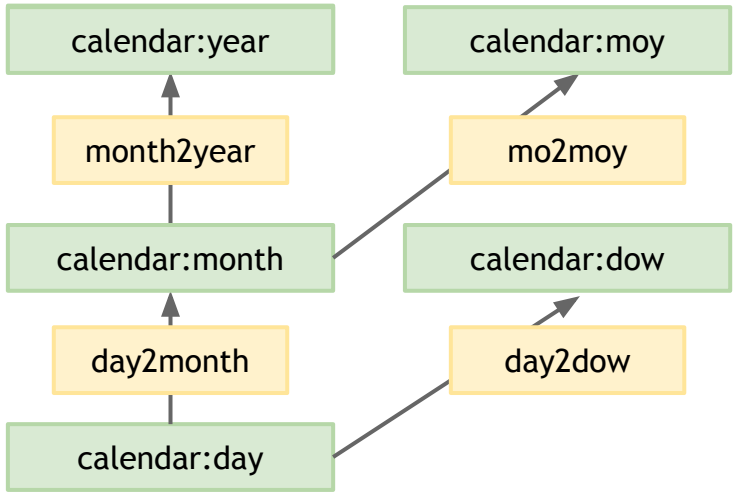




FOUNDATIONS FOR MEASURES

```
unit_sales[wine, day] = units
  -> wine(wine),
     hierarchies:calendar:day(day),
     int(units).
```





MINIMUM AGGREGATION

- minimum retail price

```
minRetail[] = minRetail
  <- agg<< minRetail = min(retail) >>
     retail[_] = retail.
```

- SQL analogy

```
SELECT min(price)
FROM retail
```

MAXIMUM AGGREGATION

- maximum margin

```
maxMargin[] = maxMargin
  <- agg<< maxMargin = max(margin) >>
     margin[_] = margin.
```

- SQL analogy

```
SELECT max(margin)
FROM margin
```

LESSON 6 LAB EXERCISES

THANK YOU.