Declarative Thinking in SQL, Its Teaching And Its Unused Potential

Declarative Amsterdam 2022

Günter Burgstaller

HTBLuVA Wiener Neustadt

November 8, 2022

Outline

1 Introduction

2 Teaching SQL as a declarative language

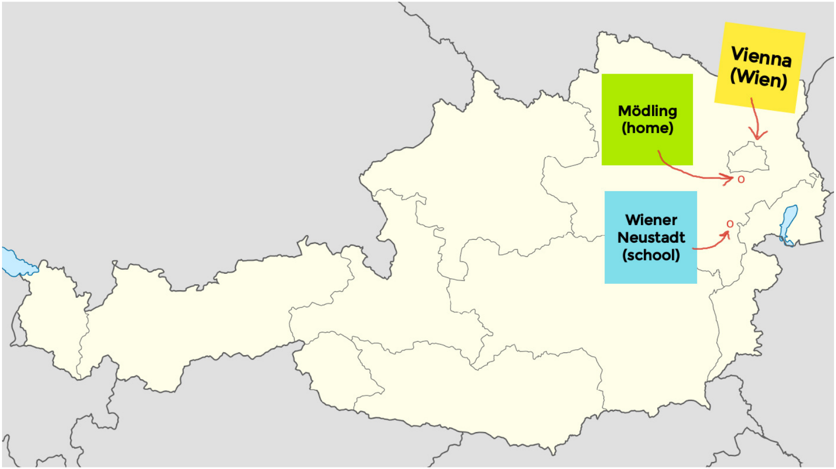
3 Case studies: declarative SQL vs. procedural Code

4 Conclusion

Austria's Technical College System (HTL)

- Students aged 14 to 19
- 5 years of practical and theoretical training in engineering
- About 34 lessons per week
- From 3rd grade on 72 lessons per year in English
- 8 weeks of mandatory summer internship
- Final exams include a diploma thesis
- Sought-after engineers for any IT profession
- Graduates are eligible for university
- Lowest tertiary level in EU

Location of HTBLuVA Wiener Neustadt



My school: HTBLuVA Wiener Neustadt

Informatics branch: what students learn

- General subjects (German, English, Maths, Science, Geography, . . .)
- Computer Architecture & Operating Systems
- Programming & Software Engineering
- Databases & Information Systems
- Network Systems & Cyber Security
- Web Programming & Mobile Computing
- Data Science & Artificial Intelligence
- Business Economics & Management
- System Planning & Project Development

Formal languages students learn

DBI Database & Information Systems

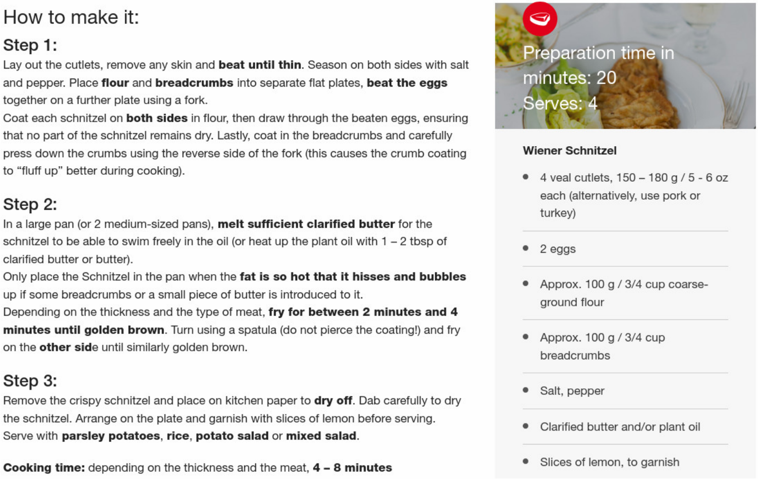
POS Programming & Software Engineering

Declarative languages are in **bold**.

Grade	POS	DBI
1	Python	(**Excel**)
2	C++	**HTML**, PHP, (**SQL**)
3	Java	**SQL**, **EBNF**, **ERD**, **XML**
4	C#, JS	**SQL**, T-SQL
5	Java, C#	**SQL**, PL/SQL, (**LaTeX**)

A simplistic explanation: Declarative Wiener Schnitzel

A simplistic explanation: Procedural Wiener Schnitzel



How to make it:

Step 1:

Lay out the cutlets, remove any skin and **beat until thin**. Season on both sides with salt and pepper. Place **flour** and **breadcrumbs** into separate flat plates, **beat the eggs** together on a further plate using a fork.

Coat each schnitzel on **both sides** in flour, then draw through the beaten eggs, ensuring that no part of the schnitzel remains dry. Lastly, coat in the breadcrumbs and carefully press down the crumbs using the reverse side of the fork (this causes the crumb coating to “fluff up” better during cooking).

Step 2:

In a large pan (or 2 medium-sized pans), **melt sufficient clarified butter** for the schnitzel to be able to swim freely in the oil (or heat up the plant oil with 1 – 2 tbsp of clarified butter or butter).

Only place the Schnitzel in the pan when the **fat is so hot that it hisses and bubbles** up if some breadcrumbs or a small piece of butter is introduced to it.

Depending on the thickness and the type of meat, **fry for between 2 minutes and 4 minutes until golden brown**. Turn using a spatula (do not pierce the coating!) and fry on the **other sid**e until similarly golden brown.

Step 3:

Remove the crispy schnitzel and place on kitchen paper to **dry off**. Dab carefully to dry the schnitzel. Arrange on the plate and garnish with slices of lemon before serving.

Serve with **parsley potatoes**, **rice**, **potato salad** or **mixed salad**.

Cooking time: depending on the thickness and the meat, **4 – 8 minutes**

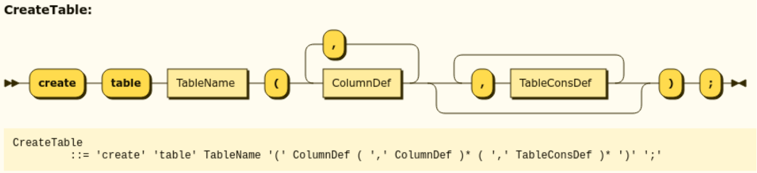
Preparation time in minutes: 20
Serves: 4

Wiener Schnitzel

- 4 veal cutlets, 150 – 180 g / 5 - 6 oz each (alternatively, use pork or turkey)
- 2 eggs
- Approx. 100 g / 3/4 cup coarse-ground flour
- Approx. 100 g / 3/4 cup breadcrumbs
- Salt, pepper
- Clarified butter and/or plant oil
- Slices of lemon, to garnish

Declarative topics in 3rd grade

Topic	Weeks
EBNF (SQL syntax)	2
SQL (DDL, DML, DQL, TCL)	24
ERD (and SQL)	10
XML (Validation, XQuery)	6



```
CreateTable
         ::= 'create' 'table' TableName '(' ColumnDef ( ',' ColumnDef )* ( ',' TableConsDef )* ')' ';'
```

Procedural extensions: relapsing into the loop obsession

Introducing T-SQL in 4th grade: why do we need procedural extensions?

- Are there problems that cannot be solved declaratively?
  - Increasing every second price
  - Increasing prices till a condition is met
- For other reasons?
- Do we need them at all?

Challenge: Increasing every second price

Increase every second price of table Parts by 5% in the order of PartID (with possible gaps e.g. row T2 missing).

```
+--------+----------+-----------+-----------+----------+
| PartID | PartName | PartColor | PartPrice | PartCity |
+--------+----------+-----------+-----------+----------+
| T1     | Mutter   | rot       | 12        | London   |
| T2     | Bolzen   | gelb      | 17        | Paris    |
| T3     | Schraube | blau      | 17        | Rom      |
| T4     | Schraube | rot       | 14        | London   |
| T5     | Welle    | blau      | 12        | Paris    |
| T6     | Zahnrad  | rot       | 19        | London   |
+--------+----------+-----------+-----------+----------+
```

Procedural: T-SQL using cursor (bad!)

```
declare _Cursor cursor        -- Cursors are slow and inefficient
for select PartID, PartPrice
      from Parts
;
open _Cursor;                 -- Error-prone due to lots of housekeeping
fetch _Cursor into @PartID, @PartPrice;
while @@fetch_status = 0                  -- Risk of inifinite loops
begin
    update Parts
       set PartPrice = @PartPrice * 1.05
     where current of _Cursor;

    fetch next from _Cursor into @PartID, @PartPrice; -- Skip a row
    fetch next from _Cursor into @PartID, @PartPrice;
end;
close _Cursor;
deallocate _Cursor;
```

Declarative: Increasing every second price

Declarative solution in SQL (for SQL Server):

```sql
update p1
   set p1.PartPrice = p1.PartPrice * 1.05
  from Parts p1
 where (select count(*) % 2 "Is2nd" -- Is the current row's position even?
          from Parts p2
         where p2.PartID <= p1.PartID) = 0
;
```

Demo #1 with SQLite. . .

Challenge: Increasing prices till a condition is met

Increase all prices of table Parts by 5% while avg(Price) < 16:

```
+--------+----------+-----------+-----------+----------+
| PartID | PartName | PartColor | PartPrice | PartCity |
+--------+----------+-----------+-----------+----------+
| T1     | Mutter   | rot       | 12        | London   |
| T2     | Bolzen   | gelb      | 17        | Paris    |
| T3     | Schraube | blau      | 17        | Rom      |
| T4     | Schraube | rot       | 14        | London   |
| T5     | Welle    | blau      | 12        | Paris    |
| T6     | Zahnrad  | rot       | 19        | London   |
+--------+----------+-----------+-----------+----------+
```

Procedural: Increasing prices till a condition is met

WHILE-loop in T-SQL to solve the problem:

```
while (select avg(PartPrice) "AvgPrice"
         from Parts) < 16
begin
  update Parts
     set PartPrice = PartPrice * 1.05
  ;
end;
```

Demo #2 with SQL Server. . .

Is there a declarative solution in SQL?

Another declarative language to the rescue:

$$avg(PartPrice) * 1.05^n = 16 \tag{1}$$

$$1.05^n = \frac{16}{avg(PartPrice)} \tag{2}$$

$$n * log(1.05) = log(\frac{16}{avg(PartPrice)}) \tag{3}$$

$$n = log(\frac{16}{avg(PartPrice)})/log(1.05) \tag{4}$$

Challenge: Increasing prices till a condition is met

Declarative solution in SQL:

```
update Parts
   set PartPrice = PartPrice
                 * power(1.05
                        , (select ceiling(log(16 / avg(PartPrice))
                                         / log(1.05)) "#"
                             from Parts)
                        )
;
```

Demo #3 with SQL Server. . .

SQL warts and wrinkles

- SQL Server: different results of procedural and declarative solution?
- Odd rounding behaviour of power-function in SQL Server
- Works fine in SQLite.

One possible fix:

```sql
update Parts
   set PartPrice = PartPrice
                 * power(1.0500000
                        , (select ceiling(log(16 / avg(PartPrice))
                                         / log(1.05)) "#"
                             from Parts)
                        )
;
```

Which RDBMS we use(d)

Which:

- SQLite
- Microsoft SQL Server Express
- Oracle Database Server XE
- (MySQL)
- (PostgreSQL)

Why?

- Employability
- Adherence to SQL ISO-Standard
- Documentation: Availability and Quality
- Ease of use (Licensing, Administration, &c.)

Task: Sieve of Eratosthenes

1. Read the Wikipedia article on the *Sieve of Eratosthenes*.
2. Fill a table `primes` with the values from 2 to 1000.
3. Write code that removes all non-primes from the table by using the sieve algorithm (consider optimisations).
4. Record the timings for your program for 1.000, 10.000, 100.000 and 1.000.000.
5. Determine the number of primes and the number of palindromic primes (e.g. 13931 is one).

Sieve of Eratosthenes sample output

Sample output for 100.000:

```
Prime numbers <= 100.000
------------------------


Elapsed: 00:00:01.71

# primes <= 10^5
----------------
            9592

# palindromic primes <= 10^5
----------------------------
                         113
```

Translating the Sieve of Eratosthenes into SQL

```
-- Delete all numbers divisible by 2 (first prime)

    delete from primes
     where mod(v_value, 2) = 0   -- mod() in Oracle, not %
    ;

-- Find next prime (which is 3)

    select min(v_value)
      from primes
     where v_value > 2
    ;

-- Repeat with 3 etc. until sqrt(1000)
```

Mostly declarative solution PL/SQL

```
declare
  next_prime integer := 2;
  max_root   integer := 0;
begin
  select nvl(sqrt(max(v_value)), 0) into max_root -- Calculate loop endpoint
    from primes
  ;

  while next_prime <= max_root loop
    delete from primes
     where v_value >= next_prime * next_prime -- Ignore, all primes already
       and mod(v_value, next_prime) = 0       -- Remove numbers divisible
    ;

    select min(v_value) into next_prime -- Find next prime as divisor
      from primes
     where v_value > next_prime
    ;
  end loop;
end;
```

Mostly procedural solution PL/SQL (bad!)

Not contrived, actually submitted:

```
declare
    primeSize integer := 1000000;
    tmp integer;

    cursor prime_cursor is       -- Slow and unnecessary
        select * from primes;
begin
    for prime in prime_cursor loop             -- Nested loops
        tmp := prime.v_value * prime.v_value;
        while tmp <= primeSize loop
            delete from primes where v_value = tmp; -- Single-row delete
            tmp := tmp + prime.v_value;
        end loop;
    end loop;
    commit;
end;
```

Using recursion in SQL?

SQL has recursion and is Turing-complete.

```
with recursive
  num(x) as (
    values(1)
    union all
    select x + 1
      from num
     where x < 1000000
)
select x
  from num
;
```

SQL: A declarative success story?

- Implementations around since 1979
- Query language for (almost) all RDBMS
- SQL is an ISO standard
  - SQL-86: 120 pages
  - SQL:2019: 15 parts, thousands of pages
- The standard is ambiguous and contradictory
- Implementations: superset of a subset of the standard

Why would we prefer declarative solutions?

- Trust the Query Optimizer: they are faster!
- Some RDBMS have no procedural extension
- More portable
- Less error-prone, e.g. no infinite loops
- Direct translation of the problem description
- Tendency to be shorter
- Easier to read? Not sure.

A programming facility is necessary

- SQLite doesn't have it
- Lack leads to code repetition
- Not necessarily procedural constructs for:
  - User defined functions (UDF)
  - Table-valued functions (TVF): Parameterized Views

Declarative: a better explanation?

Declarative Code

The abstractions that remain after generalizable, error-prone expert knowledge is moved into a reusable blackbox component.

Expert knowledge in SQL databases not to be bothered with:

- Query optimizer (Execution Plan, Query Rewrite, Query Caching, Access Paths, Join Methods, Sorting)
- Memory Management (Data Caching)
- Automatic indexing
- Constraint checking
- Transaction Management
- Concurrency Control

Omissions (maybe next time?)

Unused potential of the Relational Database Model (Chris Date: "No one has built a relational database yet.")

- SQL databases are not a faithful implementation of the Relational Database Model
- Declarative constraints (e.g. foreign keys)
- General user-defined table-/database-wide constraints not implemented
- Triggers as a poor substitute instead
- Tutorial D: a truly relational database language (Chris Date, Hugh Darwen)
  - Set behaviour of relations
  - Real user-defined data types
  - General user-definded constraints
- Rel (an experimental implementation) vs. SQLite

Literature

- Chris Date: "Relational Theory for Computer Professionals", O'Reilly, 2013.
- Chris Date: "SQL and Relational Theory", O'Reilly, 2015.
- Lex de Haan, Toon Koppelaars: "Applied Mathematics for Database Professionals", Apress, 2014.

Thank you! Questions?

Erasmus+

Enriching lives, opening minds.

Contact

```
 -----------------------------------
/ Dipl.-Ing. Prof. Günter Burgstaller \
| g.burgstaller@htlwrn.ac.at          |
| +43 2622 27871-200                  |
| (Informatics staff room)            |
|                                     |
| HTL Wiener Neustadt                 |
| Abteilung Informatik                |
| Dr. Eckener-Gasse 2                 |
| 2700 Wiener Neustadt                |
| Austria                             |
|                                     |
\ www.htlwrn.ac.at                    /
 -----------------------------------
        \   ^__^
         \  (@@)\_______
            (__)\       )\/\
                ||----w |
                ||     ||
```

- Student internships?
- Diploma thesis projects?

htl wiener neustadt

Höhere technische Bundes-
Lehr- und Versuchsanstalt