Declarative Programming: Query Languages

Denis Miginsky

Language

Query languages: the class of programming languages for searching and (sometimes) manipulating data in data-base management systems (DBMSs) or any other software focused on data.

Unlike C/Java/Haskell and other “general purposes” languages, query languages are domain-specific languages (DSLs) and are not intended for the regular programming. They are typically used as inclusion into other languages instead.

SQL: Structured Query Language (pronounces as “sequel”) – standard language for searching and manipulating data in **relational** DBMS.

Software

DBMS: SQLite – tiny embedded DBMS

Language: SQL, SQLite dialect

Software: DB Browser for SQLite, https://sqlitebrowser.org/

Hello, world!

What is the capital of Russia?

Hello, world?

Is it Moscow? Really?

What about an answer for the same question asked 200 years ago?

Let’s pretend that we do not know the answer. How shall we find it?

How to ask the same question to the computer and how it shall be answered?

Manual searching

1. Go to library
2. Take a sort of geographic handbook or encyclopedia
3. Find a page with the appropriate country
   (do we really need to do it page-by-page?)
4. Get the answer

SQL: Hello, world! Finally…

Here and below the SQLite dialect of SQL will be used.

```
SELECT CAPITAL              --what do we want to get
FROM COUNTRY_HANDBOOK       --handbook to use
WHERE COUNTRY='Russia';     --conditions
```

```
>> Moscow
   Saint Petersburg
   Petrograd
   …
```

More precise query

```
SELECT CAPITAL
FROM COUNTRY_HANDBOOK
WHERE COUNTRY='Russia'
  AND 2024 BETWEEN PERIOD_START AND PERIOD_END;
  --alternative to previous line
  AND 2024>=PERIOD_START AND 2024<=PERIOD_END;
```

>> Moscow

Basic terms (not so formal)

n-tuple (tuple) – sequence of **n** elements: $(x_1, x_2, \ldots, x_n)$

If x_i is of type T_i, then the whole tuple is of type $T_1 \times T_2 \times \ldots \times T_n$

The tuple with named positions is **record** (or structure in terms of C)

Table – unordered multiset of **records** of the same type (**rows**, **entries**)

Column (field, attribute…) – a **multiset** of elements on the particular position (identified by its name) of all the rows

COUNTRY_HANDBOOK definition

```
CREATE TABLE COUNTRY_HANDBOOK(
  COUNTRY TEXT,                    --TEXT is type in SQLite
  CAPITAL TEXT,
  PERIOD_START INTEGER,            --as well as INTEGER
  PERIOD_END INTEGER
  …);


INSERT INTO COUNTRY_HANDBOOK …

INSERT INTO COUNTRY_HANDBOOK …

…
```

Wares DB structure

```
--wares classification table

CATEGORY:
   WARE TEXT
   CLASS TEXT

Ex.: ('Charcoal', 'Fuel')
```

```
--manufacturing recipes table

MANUFACTURER:
   RECIPE_ID INTEGER
   COMPANY TEXT

Ex.: (20, 'Zogenix')
```

```
--materials for recipes

MATERIAL:
   RECIPE_ID INTEGER
   WARE TEXT
   AMOUNT INTEGER

Ex.: (19, 'Water', 7)
```

```
--outputs of "recipes"

PRODUCT:
   RECIPE_ID INTEGER
   WARE TEXT
   AMOUNT INTEGER
   PRICE REAL

Ex.: (19, 'Paper', 8, 4.6)
```

SELECT statement

```
SELECT …              --projection definition, essential
FROM …                --tables(-s) section, essential
WHERE …               --selection/filter section
ORDER BY …            --ordering section
LIMIT …               --paging section
OFFSET …


GROUP BY …
HAVING …
…
```

Projection

Projection never affects the number of result rows, only the structure of rows.

```
--simple projection,
--select subset of columns

SELECT WARE
FROM CATEGORY

>> Water
   Coal
   Paper
   Water
   …
```

```
--more complex projection
--with derivative column

SELECT RECIPE_ID, WARE,
       AMOUNT*PRICE
FROM PRODUCT


>> 3, Wood, 10.8
   4, Meat, 151.5
   4, Leather, 87.0
   …
```

Selection

Selection affects the number of result rows and never affects structure of result.

It is defined as predicate on columns.

```
--simple selection

SELECT *   --no projection
FROM CATEGORY
WHERE WARE LIKE 'W%'

>> Wood, Material
   Water, Mineral
```

```
--selection with composite
--filter and projection

SELECT CAPITAL
FROM COUNTRY_HANDBOOK
WHERE COUNTRY='Russia'
  AND 2021 BETWEEN
PERIOD_START AND PERIOD_END;
```

Unique projection

Regular projection doesn't guarantee that all the result rows are unique.

```
SELECT WARE, CLASS
FROM CATEGORY
WHERE WARE LIKE 'C%'
```

```
>> Charcoal, Material
   Charcoal, Fuel
```

```
SELECT DISTINCT WARE, CLASS
FROM CATEGORY
WHERE WARE LIKE 'C%'
```

```
>> Charcoal, Material
   Charcoal, Fuel
```

```
SELECT WARE
FROM CATEGORY
WHERE WARE LIKE 'C%'
```

```
>> Charcoal
   Charcoal
```

```
SELECT DISTINCT WARE
FROM CATEGORY
WHERE WARE LIKE 'C%'
```

```
>> Charcoal
```

Basic operations

- Arithmetic: + - * / %
- String concatenation: ‘a’ || ‘b’
- Ordering and equality predicates: >, >=, <, <=, <> (!=), = (==),
  x BETWEEN y AND z, x IS NULL, x IS NOT NULL
- Set predicates: x IN (y1, y2, y3, …)
- String predicates: x LIKE y (case insensitive in SQLite, % as wildcard)
- Logical: OR, AND, NOT
- Other useful: GLOB, REGEXP, CASE

Ordering

By default the order of result rows is arbitrary. But it could be explicitly sorted.

```
--order by single column
--in reverse alphabetical
--order

SELECT DISTINCT WARE
FROM CATEGORY
ORDER BY WARE DESC;
```

```
>> Wood
   Water
   Paper
   …
```

```
--order by two columns in
--alphabetical order

SELECT WARE, CLASS
FROM CATEGORY
ORDER BY CLASS ASC, WARE ASC;
```

```
>> Drinking water, Food
   Grain, Food
   Meat, Food
   Charcoal, Fuel
   …
```

Paging

Sometimes only a part of the result is necessary, typically to visualize it page-by-page.
In this case the ordering by all the output columns (or at least the column that forms the unique projection) is recommended (however, not required)

```
--Get the first page of size 3

SELECT DISTINCT WARE
FROM CATEGORY
ORDER BY WARE ASC
LIMIT 3;
```

```
>> Charcoal
   Drinking water
   Grain
```

```
--Get the second one

SELECT DISTINCT WARE
FROM CATEGORY
ORDER BY WARE ASC
LIMIT 3
OFFSET 3;
```

```
>> Leather
   Meat
   Meat cow
```

Set and multiset operations

Each query produces sequence of records.

It also could be viewed as sequence of tuples, or **multiset** of tuples, so set/multiset operations are applicable.

The general pattern for query with set operation:

```
SELECT … FROM … WHERE …    --q1
<SET_OPERATION>
SELECT … FROM … WHERE …    --q2
```

Both queries must produce the tuples of the same type.

Set operations is SQL

- q1 INTERSECT q2 – intersection of results of two queries (as sets)
- q1 UNION q2 – union
- q1 UNION ALL q2 – union of two multisets (all other operations consider q1 and q2 as sets and produces also the set)
- q1 EXCEPT q2 – subtracts q2 from q1

Set examples

```
SELECT WARE FROM CATEGORY
WHERE WARE LIKE 'C%'
UNION
SELECT WARE FROM CATEGORY
WHERE WARE LIKE 'D%';
```

```
>> Charcoal
   Drinking water
```

```
--the equivalent query
--DISTINCT is essential

SELECT DISTINCT WARE
FROM CATEGORY
WHERE WARE LIKE 'C%'
   OR WARE LIKE 'D%'
```

```
--union of selections from
--different tables

SELECT WARE FROM PRODUCT
UNION
SELECT WARE FROM MATERIAL
```

```
--incorrect query

SELECT WARE FROM PRODUCT
UNION
SELECT WARE, CLASS
FROM CATEGORY
```

Simple aggregation

Aggregate functions acts as reduce/fold/collect in other languages.

Using aggregate functions in projection section (under SELECT) causes query to produce exactly one tuple (until GROUP BY clause is in use) and prohibits the usage of columns without other aggregate functions.

```
--surprisingly counts rows in
--CATEGORY table
SELECT COUNT(WARE) FROM CATEGORY;
```

```
>>  16
```

```
SELECT COUNT(*),
       COUNT(),
       COUNT(DISTINCT WARE),
       COUNT(DISTINCT CLASS)
FROM CATEGORY;
```

```
>> 16, 16, 12, 7
```

```
--incorrect query
SELECT COUNT(), WARE
FROM CATEGORY;
```

Notable aggregate functions

- COUNT(), COUNT(x) – counts the rows; when provided with argument, counts all the rows where argument IS NOT NULL
- MIN(num), MAX(num) – minimum and maximum of values
- SUM(num) – sum of values
- AVG(num) – average of values
- GROUP_CONCAT(str)
  GROUP_CONCAT(str, sepatator) – concatenates text values