Solutions to CS511 Homework 13

Nicholas Ikechukwu - U71641768

December 12, 2024

Exercise 1. Some questions related to database methods

Exercise: Part (a) The SQL standard provides an operation EXISTS, which can be used as an existential quantifier. For example, SELECT ... FROM ... WHERE EXISTS $< subquery >$ So to express a database, we certainly need at least first-order logic. Argue as to whether or not second -order logic or higher is needed for any SQL operations you are familiar with. Is first-order logic sufficient for all SQL operations?

Part (a). Solution:

First-Order Logic Sufficiency in SQL

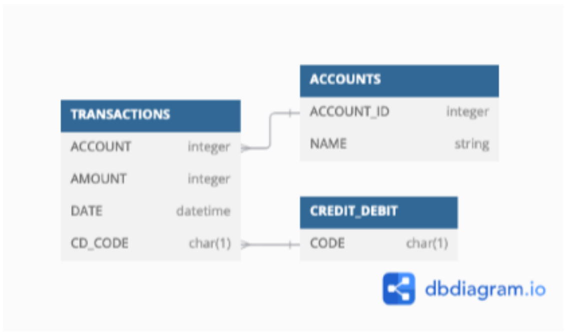
First-order logic (FOL) is sufficient for most SQL operations due to the following observations:

- The `EXISTS` operator directly corresponds to the existential quantifier $\exists$ in FOL
- Using the following, we can express SQL's core operations:
  1. Quantifiers $(\exists, \forall)$
  2. Predicates (relations)
  3. Logical connectives
- Also, when we have complex queries like joins, aggregations, and filtering, we can be construct them using FOL's expressiveness

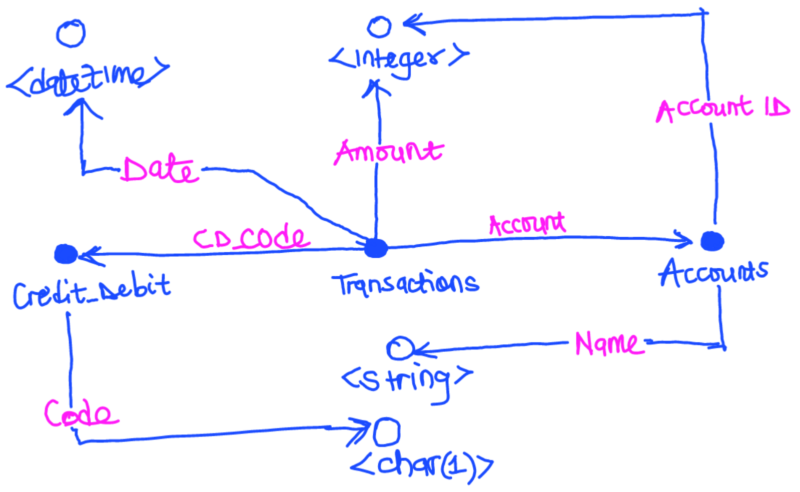
Higher-Order Logic Limitations: I believe that, although second-order logic allows quantification over predicates and relations, SQL's relational model are mostly operated on first-order. We don't actually need quantifying over predicates for standard SQL operations.

Conclusion: FOL has enough (provides a complete) logical foundation for SQL's computational model.

Exercise: Part (b) Considering the following schema: Draw a diagram representing this schema, using the diagram language from the lecture on December 3. Use filled circles for table vertices and empty circles for type vertices



Part (b). Solution



1b. Diagram representing schema

Exercise 2.

In 1763, Leonhard Euler created a very famous graph representing the islands of the Pregel River and the seven bridges across it. (To understand the very simple question he wanted to solve which motivated one of the first problems answered by graph theory, you may read the Wikipedia article on "The Seven Bridges of K¨onigsberg.") Here is the graph K which he drew, with some arrows added:

$$\begin{tikzcd}
& B \arrow[dl, bend left, "g"] \arrow[dr, "k"] & \\
A \arrow[ur, bend left, "f"] \arrow[rr, "h"] \arrow[dr, bend left, "i"] & & C \\
& D \arrow[ul, bend left, "j"] \arrow[ur, "\ell"] &
\end{tikzcd}$$

Let κ be the free category on K.

Part (a):

List the 15 morphisms of K along with their domains and codomains.

Part (a). Solution:

The free category κ on the graph K has the following morphisms:

1. Identity morphisms:
   - $id_A : A \to A$
   - $id_B : B \to B$
   - $id_C : C \to C$
   - $id_D : D \to D$
2. Basic morphisms (edges):
   - $f : A \to B$
   - $g : B \to A$
   - $h : A \to C$
   - $i : A \to D$
   - $j : D \to A$
   - $k : B \to C$

- $l : D \to C$

3. Composite morphisms:

- $f \circ g : B \to B$
- $g \circ f : A \to A$
- $h \circ f : A \to C$
- $i \circ j : D \to D$
- $l \circ j : D \to C$

These morphisms represent all possible paths and compositions in the graph K.

Part (b):

A diagram is called a "commutative" diagram if all paths with the same start and end point are equal; that is, if all "parallel" paths through the diagram produce the same result. Let κ' be the commutative free category on K; list the morphisms of κ'. Using the answers of part (a) should make this a trivial exercise.

Part (b). Solution:

Morphisms of κ'

In the commutative free category κ' on graph K, the morphisms are:

1. Identity morphisms:
   - $id_A : A \to A$
   - $id_B : B \to B$
   - $id_C : C \to C$
   - $id_D : D \to D$
2. Basic morphisms:
   - $f : A \to B$
   - $g : B \to A$
   - $h : A \to C$
   - $i : A \to D$
   - $j : D \to A$
   - $k : B \to C$
   - $l : D \to C$
3. Composite morphisms:
   - $f \circ g = id_B : B \to B$
   - $g \circ f = id_A : A \to A$
   - $k \circ f = h : A \to C$
   - $l \circ j = h : A \to C$
   - $i \circ j = id_D : D \to D$

Part (c):

Consider the following category V: U - p → V ← q - W
Define a functor F : V → κ.

Part (c). Solution:

Functor $F : V \to \kappa$

From what we have about κ, given the category V with objects U, V, W and morphisms $p : U \to V$, $q : W \to V$, we can define the functor $F : V \to \kappa$ as follows:

- **Mapping of Object :**

$$F(U) = A$$
$$F(V) = B$$
$$F(W) = D$$

- **Mapping of Morphism :**

$$F(p) = f$$
$$F(q) = l$$

This shows that the functor maps the objects and morphisms of V to the corresponding objects and morphisms in the category we have, κ.

Part (d):

Imagine a category that looks like this:

$$\begin{tikzcd}
\bullet \arrow[r] & \bullet \arrow[d] \\
\bullet \arrow[u] & \bullet \arrow[l]
\end{tikzcd}$$

Argue why there cannot be a functor from this category to κ.

Part (d). Solution:

My Simple Argument Against a Functor to κ

From the category given above, it forms forms a cycle with four objects and morphisms creating a loop. We know that In category theory:

- The category κ, as we had, does not have a structure that supports cyclic morphisms.
- A functor must map objects and morphisms in a way that preserves composition and identity.
- Particularly, κ lacks the necessary morphisms to map a cycle of four objects back onto itself.

Therefore, we cannot have a functor from this cyclic category to κ, as it would violate the necessities to preserve composition and identity in the presence of cycles.

PROBLEM 1. Adjunction Between Functors

Exercise :

The preceding example about currying illustrates of what is called an **adjunction** between functors, here the functor $- \times B$ and the functor $(-)^B$. We only said how each of these two functors works on objects: For an arbitrary set X, the first functor returns the set $X \times B$ while the second returns the set X^B.
There are three parts in this problem – these may look scary to you, but the answer to each part takes at most 2 (or perhaps 3) lines:

1. Given a morphism $f : X \to Y$, what morphism should $- \times B : X \times B \to Y \times B$ return?
2. Given a morphism $f : X \to Y$, what morphism should $(-)^B : X^B \to Y^B$ return?
3. Consider the function $+ : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, which maps (a, b) to $a + b$. Currying $+$ we get a function $p : \mathbb{N} \to \mathbb{N}^{\mathbb{N}}$. What is $p(3)$?

Part 1:

Given a morphism $f : X \to Y$, what morphism should $- \times B : X \times B \to Y \times B$ return?

Part (1). Solution: The Morphism for $- \times B$

Since we have $f : X \to Y$, the functor $- \times B$ should give the morphism:

$$f \times id_B : X \times B \to Y \times B$$

Where id_B denotes the identity morphism on B, and $f \times id_B$ is defined as:

$$(f \times id_B)(x, b) = (f(x), b)$$

Part 2:

Given a morphism $f : X \to Y$, what morphism should $(-)^B : X^B \to Y^B$ return?

Part (2). Solution: The Morphism for $(-)^B$

We have a morphism $f : X \to Y$, the functor $(-)^B$ should give the morphism:

$$f^B : X^B \to Y^B$$

this is defined pointwise as:

$$(f^B)(g)(b) = f(g(b))$$

Where $g \in X^B$ and $b \in B$

Part 3:

Consider the function $+ : \mathbb{N} \times \mathbb{N} \to \mathbb{N}$, which maps (a, b) to $a + b$. Currying $+$ we get a function $p : \mathbb{N} \to \mathbb{N}^{\mathbb{N}}$. What is $p(3)$?

Part (3). Solution: Currying Addition

Looking at the function, we should have that $p(3)$ is the function:

$$p(3) : \mathbb{N} \to \mathbb{N}$$

This is defined as:

$$p(3)(b) = 3 + b$$

More clearly but comprehensively:

$$p(3) = \lambda b.3 + b$$

ON LEAN-4

Solutions in one file at: https://github.com/nich-ikech/CS511-hw-macbeth/blob/main/cs511HwSolutions/hw13/hw13_nicholas_ikechukwu.lean

Exercise 3. From Macbeth's book: Exercise 10.1.5.4

Solutions

https://github.com/nich-ikech/CS511-hw-macbeth/blob/main/cs511HwSolutions/hw13/hw13_nicholas_ikechukwu.lean

Exercise 4. From Macbeth's book: Exercise 10.1.5.5

Solutions

https://github.com/nich-ikech/CS511-hw-macbeth/blob/main/cs511HwSolutions/hw13/hw13_nicholas_ikechukwu.lean

PROBLEM 2. From Macbeth's book: Exercise 10.1.5.6

Solutions

https://github.com/nich-ikech/CS511-hw-macbeth/blob/main/cs511HwSolutions/hw13/hw13_nicholas_ikechukwu.lean