Practical Assignment of Advanced Topics in Databases

Lourenço Pedro Magalhães Giuseppe

This report summarizes the design and implementation choices made during the "Practical Assignment: Advanced Topics in Databases" project. It provides a detailed overview of the database setup, data modeling decisions, and initial data population strategy, emphasizing the rationale behind key design choices.

Database Setup and Data Modeling

For the core database technology, PostgreSQL was selected primarily due to its robust and highly performant support for spatial data through the PostGIS extension. This was a critical factor given the geometric nature of the problem domain.

The following Entity-Relationship (ER) diagram (Figure 1) illustrates the implemented schema and its structural relationships:

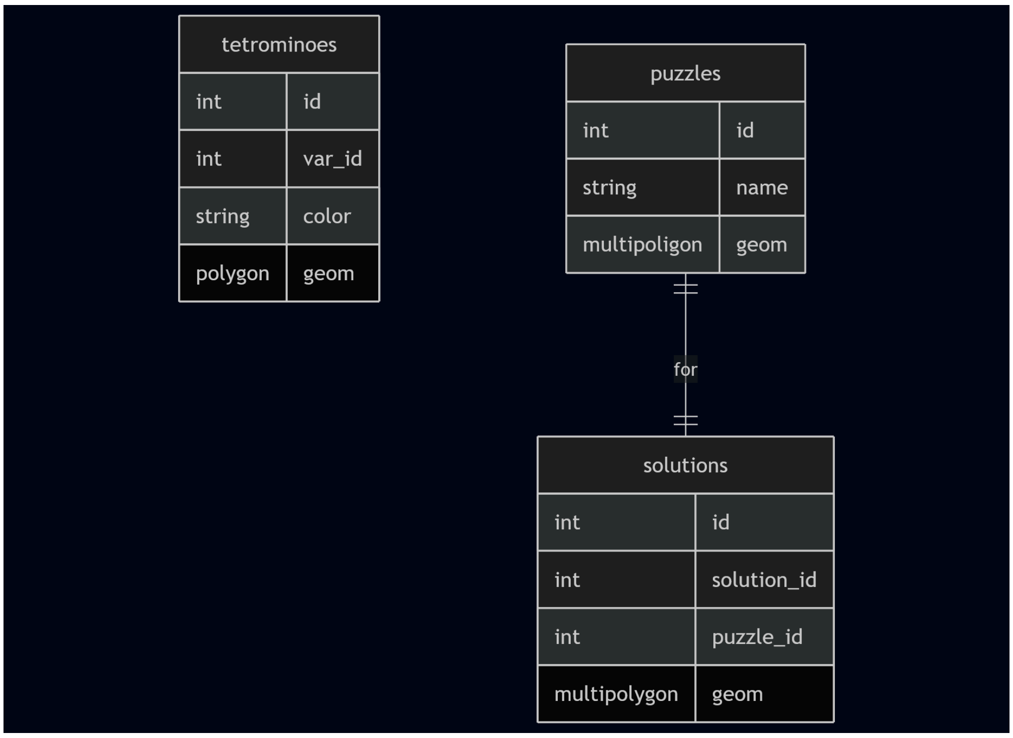


Figure 1: fig1: Entity-Relationship Diagram of the Database Schema

A significant design decision involved the representation of individual "solutions." Each solution was modeled as a composite of pre-defined tetromino configurations, each already possessing a specific rotation and spatial location. This was efficiently represented using a MULTIPOLYGON spatial data type within PostGIS.

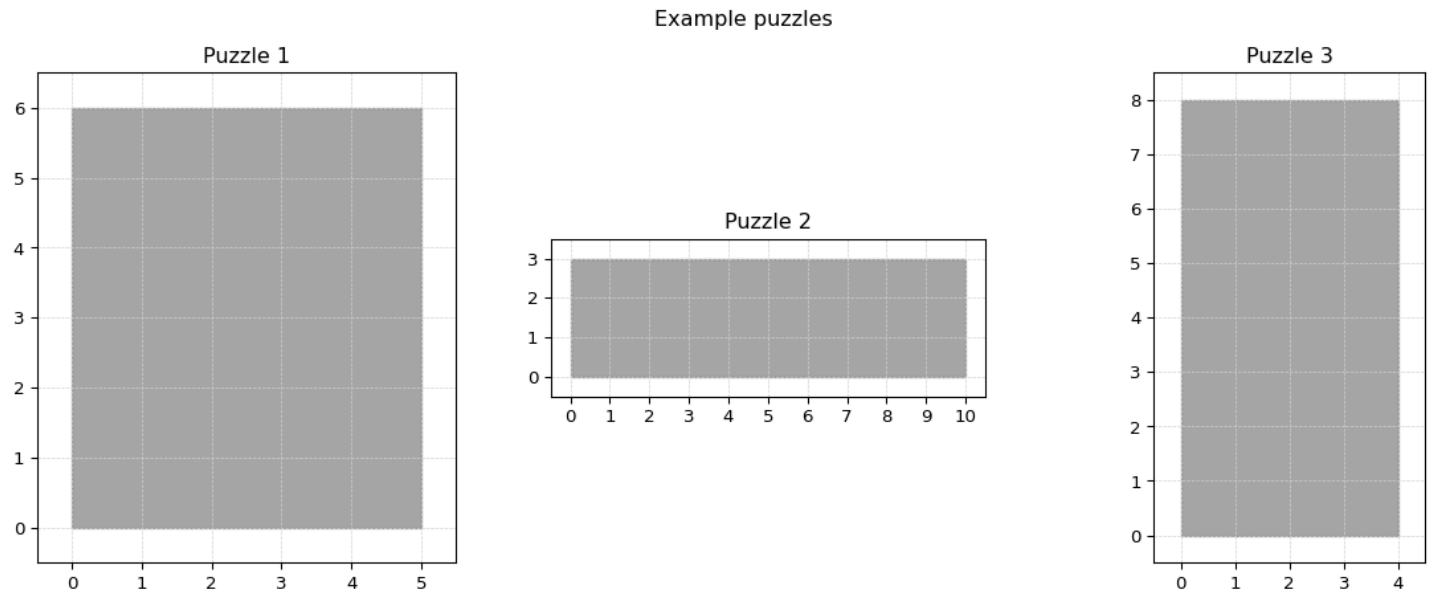
Initially, a more normalized database structure, adhering to the Third Normal Form (3NF), was considered. This would have entailed a bridge table between tetrominoes and solutions to explicitly link individual tetromino instances to complete solutions. Such a normalized approach would undeniably enhance data model consistency and enforce stricter adherence to a predefined set of available figures, ensuring data integrity at a more granular level. However, within the defined scope and constraints of this project, it was determined that the additional complexity introduced by this normalization would not yield proportional benefits in terms of performance or problem-solving efficiency for the solver component. The current denormalized approach was chosen to streamline data access and reduce query complexity for the specific needs of the solver's operational workflow.

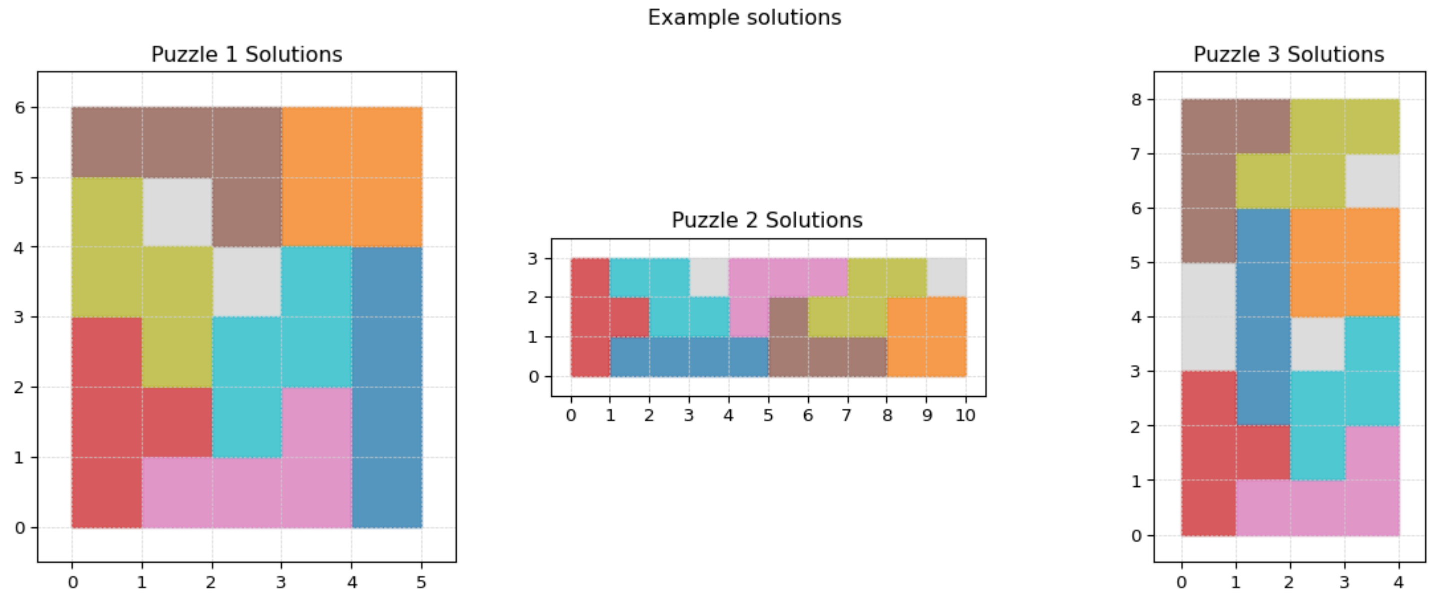
Furthermore, a deliberate choice was made to pre-represent all nineteen (19) distinct rotational and positional variations of each of the seven fundamental tetrominoes, as opposed to storing only the base seven shapes. This strategic decision offered several significant advantages:

- **Elimination of Runtime Rotation Calculations:** By pre-computing and storing all variations, the need to perform computationally intensive geometric rotation functions within the solver at runtime was entirely obviated. This directly contributes to improved solver performance and reduced computational overhead.
- **Predictable Initial Positioning:** This approach allowed for the precise prediction of the exact starting position for each pre-configured tetromino piece within a solution. This predictability significantly reduced the search space and the number of permutations the solver needed to evaluate, thereby minimizing backtracking and iteration cycles. This optimization was crucial for enhancing the efficiency of the solver's heuristic or algorithmic search process.

Initial Data Population

To facilitate testing and validation of the implemented solver, the database environment was populated with three distinct puzzle configurations and their corresponding verified solutions. These pre-defined data sets serve as crucial benchmarks and guidance for the subsequent development and rigorous testing of the solver's functionality and accuracy.





Solver Integration and Predicate Implementation

The solver's core operations—transposing tetrominoes, detecting intersections, and confirming boundary adherence—were directly mapped to Prolog predicates backed by PostGIS spatial functions.

Solver Integration and Predicate Implementation

The solver's core operations—transposing tetrominoes, detecting intersections, and confirming boundary adherence—were directly mapped to Prolog predicates backed by PostGIS spatial functions. This approach leverages Prolog's powerful declarative reasoning capabilities for search and backtracking, while offloading computationally intensive geometric operations to the highly optimized PostGIS engine.

Specifically, the following functionalities were implemented as predicates:

- **Intersection Detection:** To determine if a newly positioned tetromino overlaps with any previously placed pieces, a predicate utilizing the `ST_Touches` PostGIS function was employed.
- **Boundary Confinement:** To ensure that a tetromino remains within the defined bounds of the puzzle grid after transposition, a predicate using the `ST_Within` PostGIS function was integrated. This predicate verifies that the entire geometry of the tetromino is contained within the puzzle's boundary.
- **Solution Aggregation:** As the solver successfully places tetrominoes, their geometries need to be cumulatively stored to represent the evolving solution. For this, a predicate encapsulating the `ST_Collect* PostGIS function was implemented. The first interaction the`GEOMETRYCOLLECTION EMPTY` empty group is passed.

Solver solution

TThe solver's primary objective is to find a valid arrangement of tetrominoes that completely fills the predefined puzzle space without overlaps. The solver's logic is encapsulated in Prolog predicates, which leverage the pre-processed spatial data from the database.

A key optimization involves organizing the 19 distinct tetromino variations. Rather than treating them as an undifferentiated list, they are grouped by their fundamental tetromino type (e.g., all 'L' shaped variations, all 'T' shaped variations). This is achieved by creating a YAP/Prolog atom (function YAP_Term create_tetramino_list) that transforms the flat list of all variations into a structured list of group(Letter, Variations). Once a valid placement for one variation within a group is found, the solver can avoid re-iterating over other variations of the same fundamental tetromino type in that specific branch of the search space, significantly pruning the search tree.

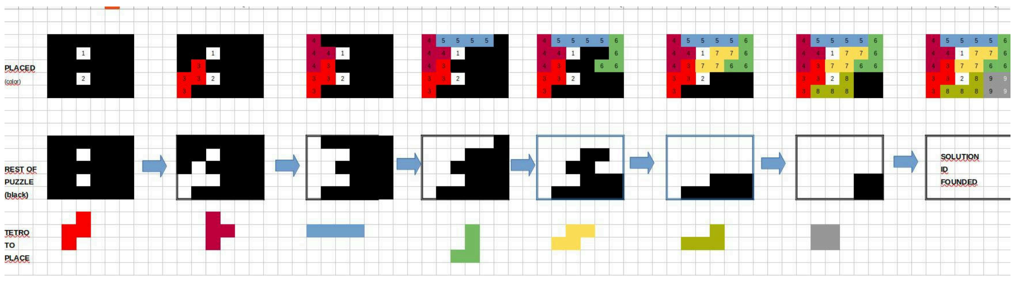


Figure 2: fig2: Solver Diagram

The solve/4 predicate orchestrates the placement process. It recursively iterates through these group lists. For each group, it attempts to place a tetramino(Letter, Seq, TetWKT) from its Variations list.

The actual placement logic resides within the try_place/4 predicate. This predicate employs a systematic grid-based search using grid_offset/2 to generate potential (Dx, Dy) translation coordinates. For each candidate position:

1. Transposition: The transpose_geometry predicate (an external function, likely calling PostGIS's ST_Translate functionality) is used to move the current TetWKT (Well-Known Text representation of the tetromino's geometry) to the new (Dx, Dy) offset, producing CandidatePlacedTetWKT.

2. Intersection Check: The disjoint_geometry predicate (corresponding to ST_Touches in PostGIS) verifies that the CandidatePlacedTetWKT does not intersect with OccupiedGeom, which represents the accumulated geometry of all already placed tetrominoes.

3. Boundary Confinement: The within_geometry predicate (mapping to PostGIS's ST_Within function) ensures that the CandidatePlacedTetWKT is entirely contained within the Puzzle boundary.

If all these conditions are met, the CandidatePlacedTetWKT is accepted as PlacedTet, and the solve predicate recursively calls itself, updating the AccGeom (accumulated geometry) by unifying the newly placed tetromino using the union_geometry predicate. This iterative process continues until all tetromino groups are successfully placed, leading to FinalPuzzle representing the complete solution.