1. Project Discription

Our Pokemon Data Management System is designed to help trainers optimize their gameplay by allowing them to store, delete, and retrieve specific Pokemon data. In addition to basic data management, the system offers insights that support strategic decision-making. Users can view average attack values grouped by type name, identify type names with an average defense greater than 10, and find trainers who own at least one Pokémon from every category. Through these features, our system helps trainers make more informed choices and improve their overall game performance.

2. Schema Differences and Explanation

We changed the schema name from **Type** to **PokemonType** to improve readability.

3. SQL Query List and Locations in Code

Functionality	SQL Query Summary	Line(s)

Queries1: INSERT	Insert into PokemonTrains with FK check on TypeName
	`INSERT INTO PokemonTrains (PokemonID, PokemonName, TypeName, PokemonGender, Ability, TrainerID)`
	`VALUES (:id, :name, :type, :gender, :ability, :trainer)`
		appService.js line: 174 link

Query2: Update	Updates any combination of non-PK fields: PokemonName, TypeName, PokemonGender, Ability, TrainerID.
	`UPDATE PokemonTrains`
	`SET ${update.join(', ')}`
	`WHERE PokemonID = :id`
		appService.js, line: 265 link

Query3: Delete	Delete a tuple from PokemonTrains by PokemonID `DELETE FROM PokemonTrains WHERE PokemonID = :id`	appService.js, line: 287 link
Query4: Selection	Allowed to search for tuples using any number of AND/OR clauses and combinations of attributes in PokemonTrains	appService.js, line: 310 link

	`SELECT ${selectClause} FROM PokemonTrains p JOIN PokemonType t ON p.TypeName = t.TypeName WHERE ${whereClause}`;	
Query5: Projection	Project any number of attributes from the PokemonTrains table `SELECT ${attributes} FROM PokemonTrains p JOIN PokemonType t ON p.TypeName = t.TypeName`)	appService.js, line: 331 link
Query6: Join	Join Trainer and PokemonTrains to find all Pokemon trained by a specific trainer SELECT t.TrainerID, t.TrainerName, p.PokemonID, p.PokemonName FROM Trainer t JOIN PokemonTrains p ON t.TrainerID = p.TrainerID WHERE t.TrainerID=:trainerID	appService.js, line: 345 link
Query7: Aggregation With GROUP BY	Get average attack value grouped by TypeName SELECT p.TypeName, AVG(s.Attack) AS AvgAttack FROM PokemonTrains p JOIN Shows sh ON p.PokemonID = sh.PokemonID JOIN Stats s ON sh.StatsID = s.StatsID GROUP BY p.TypeName	appService.js, line 365 link
Query8: Aggregation With HAVING	Get TypeNames with average defense > 10 SELECT p.TypeName, AVG(s.Defense) AS AvgDefense FROM PokemonTrains p JOIN Shows sh ON p.PokemonID = sh.PokemonID	appService.js, line 383 link

	JOIN Stats s ON sh.StatsID = s.StatsID GROUP BY p.TypeName HAVING AVG(s.Defense) > 10 FROM PokemonTrains p JOIN Shows sh ON p.PokemonID = sh.PokemonID JOIN Stats s ON sh.StatsID = s.StatsID GROUP BY p.TypeName HAVING AVG(s.Defense) > 10	
Query9: Nested Aggregation	Get trainer names whose Pokemon have above-average total stats SELECT t.TrainerName, AVG(s.HP + s.Attack + s.Defense + s.SpecialAttack + s.SpecialDefense + s.Speed) AS AvgTotalStats FROM Trainer t JOIN PokemonTrains pt ON t.TrainerID = pt.TrainerID JOIN Shows sh ON pt.PokemonID = sh.PokemonID JOIN Stats s ON sh.StatsID = s.StatsID GROUP BY t.TrainerName HAVING AVG(s.HP + s.Attack + s.Defense + s.SpecialAttack + s.SpecialDefense + s.Speed) > (SELECT AVG(HP + Attack + Defense + SpecialAttack + SpecialDefense + Speed) FROM Stats)	appService.js, line: 402 link
Query10:Division	Get trainers who own at least one Pokemon from every category SELECT t.TrainerName FROM Trainer t WHERE NOT EXISTS (SELECT c.CategoryName FROM Category c MINUS SELECT b.CategoryName FROM BelongsTo b JOIN PokemonTrains pt ON b.PokemonID = pt.PokemonID	appService.js, line: 424 link

	WHERE pt.TrainerID = t.TrainerID)	

4. Acknowledgment

We used Grammarly to improve sentence fluency and correct syntax mistakes.

We based the initial structure of our project on the CPSC304 provided sample project.