**1. Concrete Table Inheritance (Database: concreatetable)**

In this approach, each type of employee has its own table (empx, reg\_empx, cont\_empx):

* **Tables:**
  + empx: Stores general employee details (e.g., ID, first name, last name).
  + reg\_empx: Stores details specific to regular employees (e.g., bonus, salary).
  + cont\_empx: Stores details specific to contract employees (e.g., number of periods, rate per hour).
* **Observations:**
  + The id column acts as the primary key and links records across these tables.
  + Queries for specific employee types must join the general table (empx) with the relevant subtype table (reg\_empx or cont\_empx).
  + Separation of data reduces redundancy but requires multiple queries or joins to retrieve complete employee details.

**2. Separate Table Inheritance (Database: SeperateTable)**

In this approach, each type of employee is stored in a separate table, with duplication of common fields:

* **Tables:**
  + emp2: Stores generic employee information, similar to empx in the first design.
  + contractemployee: Contains both general (fname, LastName) and specific fields (No\_periods, rate\_per\_hr) for contract employees.
  + regularemployee: Contains both general (fname, LastName) and specific fields (bonus, salary) for regular employees.
* **Observations:**
  + Each employee type is stored in a distinct table, including shared attributes like fname and LastName.
  + Data is easier to query for a specific employee type (no need for joins).
  + Data redundancy increases because common attributes are duplicated across tables.

**3. Single Table Inheritance (Database: singletable)**

This approach consolidates all employee types into one table, with columns for all possible fields:

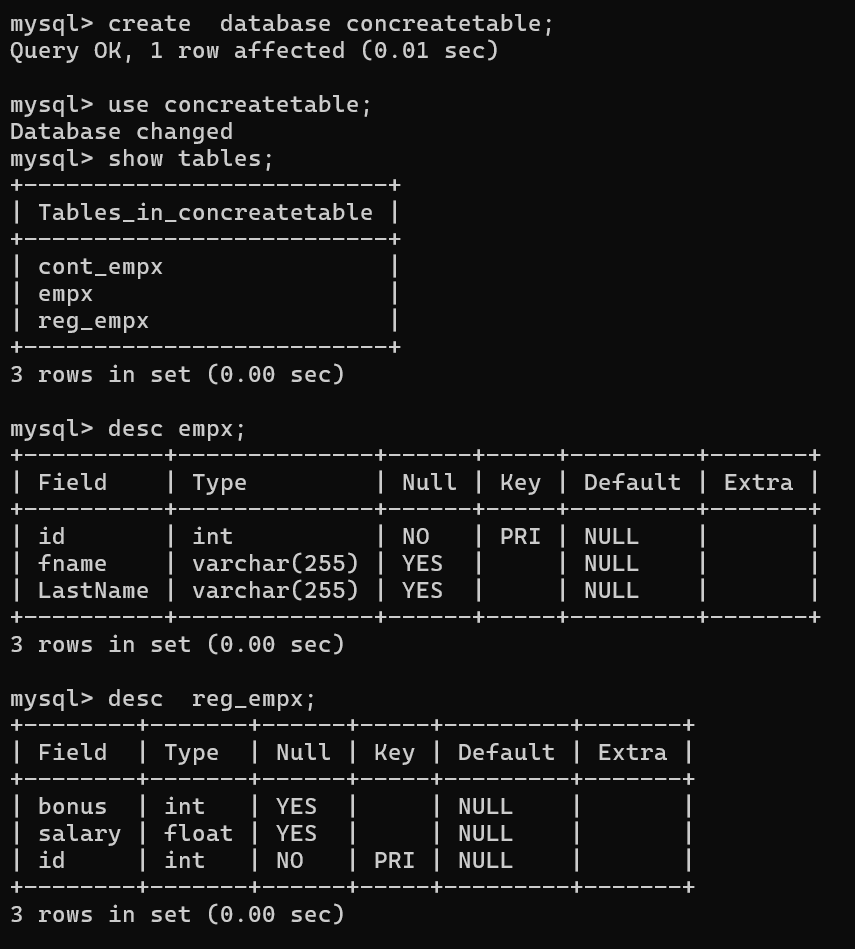
* **Table:**
  + emp: A single table storing all employee types, identified by the Employee\_type column.
  + Fields not relevant to a specific employee type (e.g., bonus for a contract employee) are left NULL.
* **Observations:**
  + Simplifies querying, as all data is stored in one place.
  + Space inefficiency arises because of many NULL values for unused fields.
  + Adding new employee types or attributes requires altering the table structure.

**Comparison of Approaches**

| **Feature** | **Concrete Table Inheritance** | **Separate Table Inheritance** | **Single Table Inheritance** |
| --- | --- | --- | --- |
| **Ease of Querying** | Medium (requires joins) | High (separate tables) | High (single table) |
| **Data Redundancy** | Low | High | Medium |
| **Space Efficiency** | High | Medium | Low (due to NULLs) |
| **Ease of Adding Attributes** | Medium | High (separate tables) | Low (table alteration) |

**Key Takeaways:**

* **Concrete Table Inheritance** works well when attributes are largely distinct across types and querying for specific types is common.
* **Separate Table Inheritance** is suitable when the types are entirely independent and do not need to share many relationships.
* **Single Table Inheritance** is effective for simplicity when the variations between types are minimal.



A screenshot of a computer program

Description automatically generated

A screenshot of a computer program

Description automatically generated

A screenshot of a computer

Description automatically generated