1. **Jaka rola klasy Contact i z jaką tabelą jest powiązana?**

`Contact` to **model danych** (encja) reprezentujący pojedynczy rekord kontaktu w aplikacji. Jest powiązana z tabelą bazy danych przechowującą kontakty (zwykle o nazwie **Contacts** albo **Contact**) – jej pola odpowiadają kolumnom w tej tabeli (np. `Id`, `FirstName`, `LastName`, `Phone`, `Email`).

2. **Dlaczego Phone i Email są opcjonalne (string?)?**
   Bo **nie każdy kontakt musi mieć telefon i/lub e-mail**. `string?` oznacza, że wartość może być `null`, czyli pole jest **opcjonalne** (np. użytkownik poda tylko imię i nazwisko). To też pasuje do bazy, gdzie kolumny `Phone`/`Email` mogą dopuszczać `NULL`.

3. **Po co architektura warstwowa (Model / DAL / Program)?**
   Żeby rozdzielić odpowiedzialności:

- **Model**: same dane i proste zasady (np. `Contact`).
- **DAL (Data Access Layer)**: logika dostępu do bazy (SQL, połączenia, mapowanie wyników).
- **Program/UI**: obsługa menu, wejścia/wyjścia, wywoływanie metod DAL.
  To daje czytelność, łatwiejsze testowanie, łatwiejsze zmiany (np. podmiana bazy) i mniej "bałaganu" w `Program.cs`.

4. **Jaka odpowiedzialność ContactRepository i dlaczego tam jest SQL?**

`ContactRepository` odpowiada za **CRUD** i operacje na danych kontaktów w bazie (Create/Read/Update/Delete, wyszukiwanie, bulk insert itd.). SQL jest w repozytorium, bo to warstwa DAL – miejsce,

gdzie **powinien być kod dostępu do danych**, a nie w UI.

5. **Co to connectionString i jakie 2 kluczowe informacje musi zawierać?**

`connectionString` to tekst konfiguracyjny mówiący aplikacji **jak połączyć się z bazą**. Musi zawierać kluczowo:

- **adres/instancję serwera** (np. `Server=...` / `Data Source=...`)
- **nazwę bazy danych** (np. `Database=...` / `Initial Catalog=...`)
  Dodatkowo zwykle zawiera sposób uwierzytelniania (Windows Auth lub login/hasło).

6. **Jakie klasy ADO.NET są używane do komunikacji z SQL Server?**
   Najczęściej:

- `SqlConnection` – połączenie z bazą
- `SqlCommand` – zapytanie/komenda SQL
- `SqlDataReader` – odczyt wyników SELECT
- `SqlParameter` (lub `command.Parameters.Add...`) – parametry zapytań
  Czasem też `SqlTransaction` – transakcje, oraz `SqlDataAdapter`/`DataTable` (jeśli użyte, choć w repozytoriach zwykle `Reader` wystarcza).

7. **Dlaczego parametry SQL (@fn, @ln…), a nie konkatenacja stringów?**

Bo parametry:

- chronią przed **SQL Injection**
- poprawnie obsługują znaki specjalne (np. apostrofy w nazwiskach)
- ułatwiają typowanie (int, nvarchar, itp.)
- mogą poprawiać wydajność (plan zapytań może być reużywany)

8. **Jaką metodę ADO.NET do INSERT/UPDATE/DELETE i dlaczego?**
   Używa się `ExecuteNonQuery()`, bo te polecenia **nie zwracają tabeli wyników**, tylko liczbę zmienionych wierszy (rows affected). To jest dokładnie to, czego potrzebujesz przy operacjach modyfikujących dane.
9. **Jak program pobiera Id nowo dodanego kontaktu po INSERT?**
   Typowe podejścia:

- `INSERT ...; SELECT SCOPE_IDENTITY();` i wykonanie `ExecuteScalar()`
  albo
- `OUTPUT INSERTED.Id` w zapytaniu `INSERT` i `ExecuteScalar()`
  W obu przypadkach baza zwraca wygenerowany identyfikator, a kod przypisuje go do obiektu `Contact`.

10. **Jak działa wyszukiwanie po nazwisku i czemu LIKE?**
    Program wykonuje `SELECT ... WHERE LastName LIKE @pattern` (np. `@pattern = "Kow%"` albo `"%Kow%"`). `LIKE`

umożliwia **wyszukiwanie częściowe** (fragment nazwiska), a nie tylko dokładne dopasowanie.

11. **Co to transakcja bazodanowa i po co w BulkInsert?**
    Transakcja to mechanizm, który grupuje wiele operacji w jedną "paczkę": **albo wszystkie się udadzą, albo żadna**. W `BulkInsert` jest po to, żeby przy dodawaniu wielu kontaktów zapewnić spójność: nie chcesz sytuacji, że połowa kontaktów się dodała, a połowa nie.

12. **Co się stanie z danymi, jeśli w BulkInsert wystąpi błąd?**
    Jeśli jest transakcja: następuje **rollback**, czyli baza **cofa wszystkie operacje** wykonane w ramach tej transakcji – finalnie **nie zapisze się nic** z tej paczki (lub wróci do stanu sprzed transakcji).

13. **Do czego służy Logger i co trafia do logs.txt?**

`Logger` służy do zapisywania informacji diagnostycznych – najczęściej:

- daty i godziny zdarzeń
- opisu operacji (np. "Dodano kontakt", "BulkInsert start/stop")
- błędów/wyjątków (treść wyjątku, stack trace jeśli dodany)
- czasem informacji o parametrach/ID (bez danych wrażliwych, zależnie od implementacji)

14. **Jak program zabezpiecza się przed niepoprawnym identyfikatorem (tekst zamiast liczby)?**
    Przez walidację wejścia: użycie `int.TryParse()` (zamiast

`int.Parse()`), pętla ponawiająca pytanie albo obsługa `FormatException`. Dzięki temu program nie wykrzacza się i wymusza poprawną liczbę.

15. **Dlaczego Program.cs nie ma bezpośrednio SQL i tylko wywołuje repozytorium?**
    Bo `Program.cs` to warstwa UI/sterowanie aplikacją. SQL należy do DAL. Taki podział:

- ułatwia utrzymanie (zmiana SQL nie dotyka UI)
- zmniejsza duplikację kodu
- poprawia czytelność
- pozwala łatwo testować logikę dostępu do danych i ewentualnie podmienić źródło danych (np. plik zamiast SQL)

Jeśli chcesz, mogę Ci te odpowiedzi od razu ułożyć w gotowy dokument `.docx` (z nagłówkiem, numeracją i krótkim wstępem) — wystarczy, że powiesz, jak ma się nazywać plik.