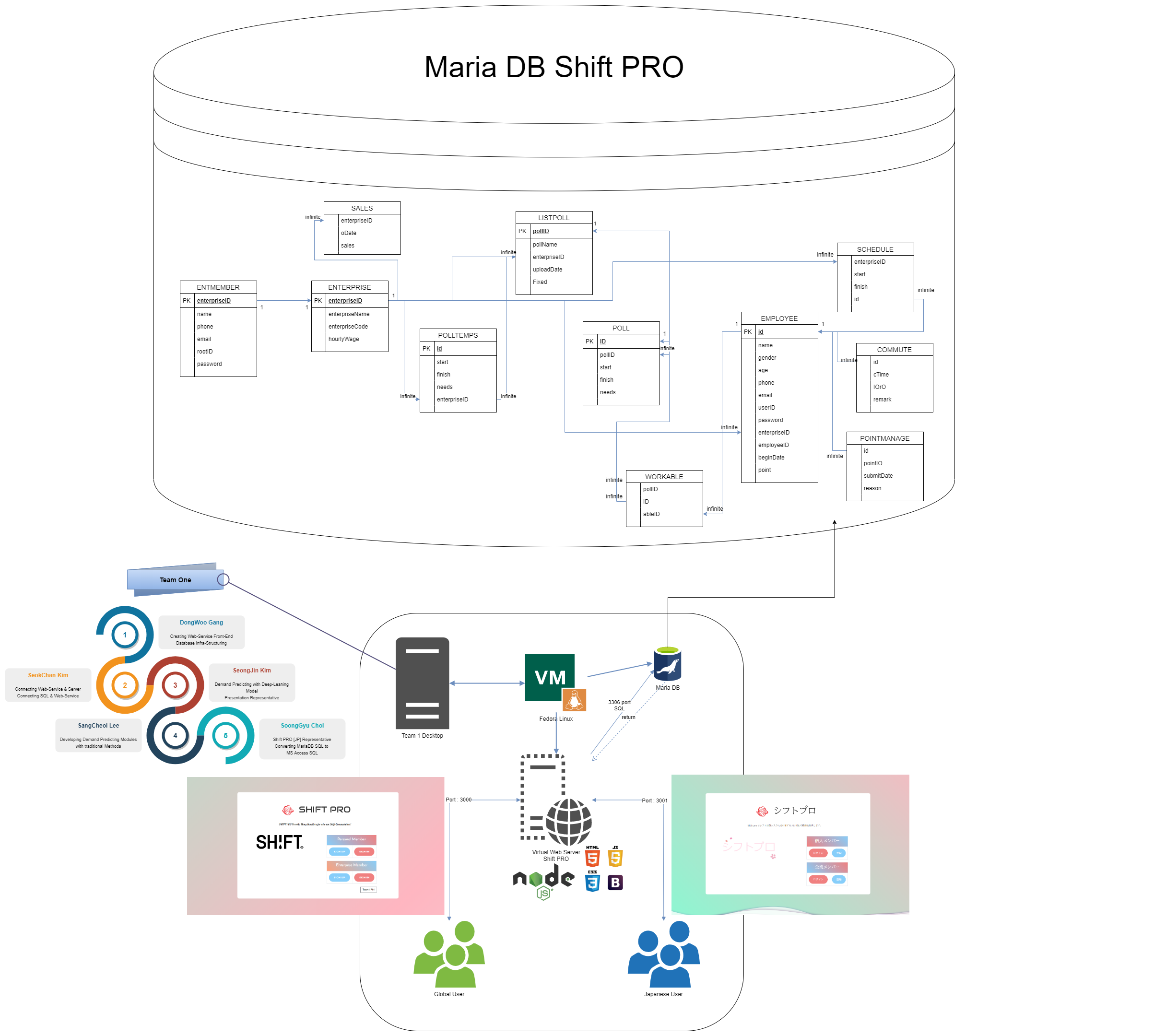
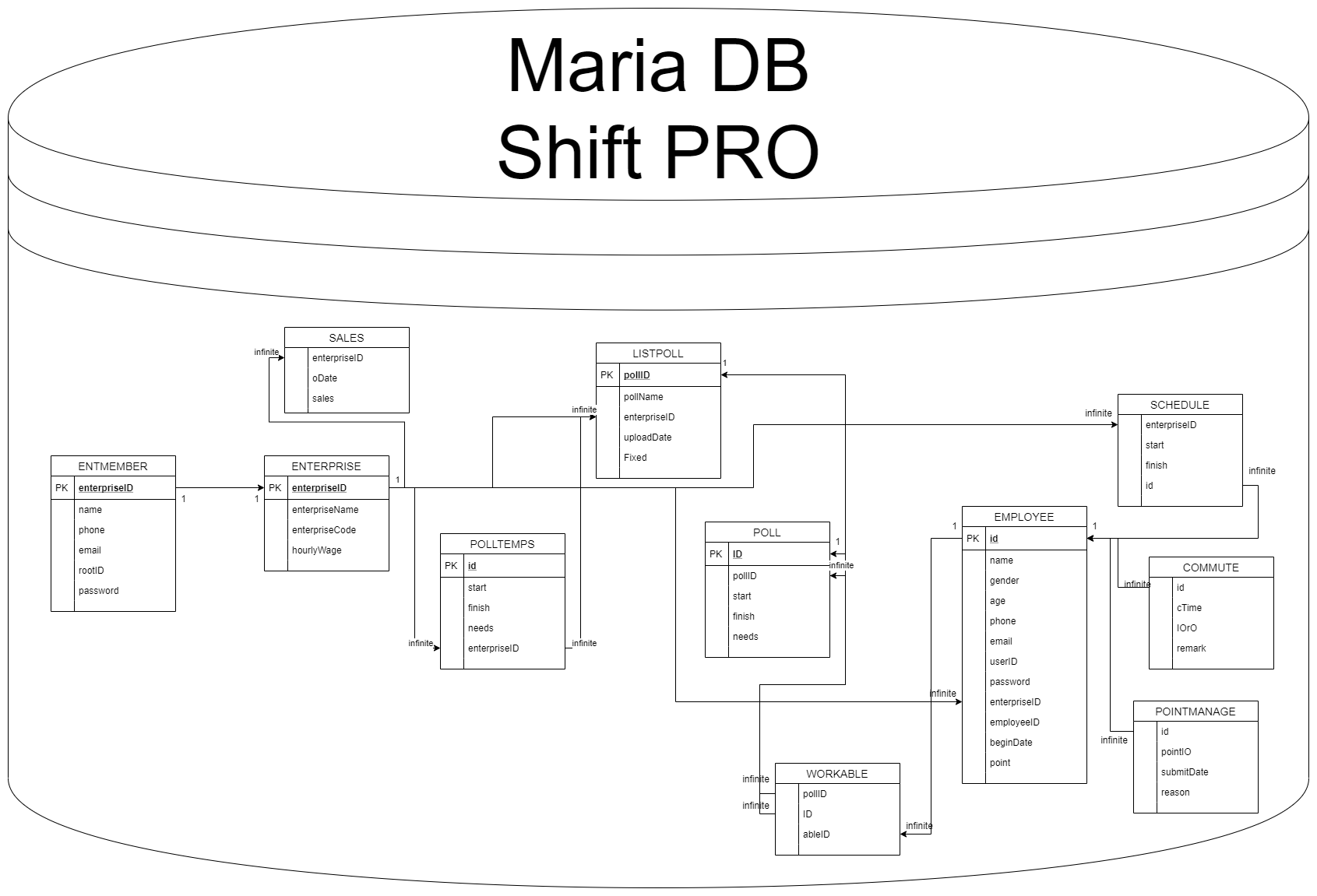
* Service Architecture



* Database Scheme



Team one DB Scheme의 장점

1. **Universal & Lean**

초창기에 데이터베이스를 설계하면서 가장 중점적으로 고려했던 부분은 위에 기술된 **“범용적이면서도 Lean한 구조”**이다. 현재 데이터베이스의 구조는 **11개의 데이터베이스를 기반으로(딥러닝 수요 예측 기반 테이블이 하나 추가된다면 12개) 수천개의 회사, 수만명의 종업원을 모두 담을 수 있다.** 데이터베이스의 구조를 이렇게 설계한 이유는 범용성이 전부가 아니다. 범용성만을 고려했다면 한 회사당 하나의 테이블, 그에 연결되는 모든 테이블들을 각 회사에 생성해주는 것으로도 구현할 수 있다.(1개의 회사가 위 11개의 테이블을 가지는 구조: 10개의 회사 = 110개의 테이블 요구) 그러나 이러한 접근은 어마어마한 낭비를 유발한다. 고작 수백개의 회사에 수천명의 종업원의 등록만으로 데이터베이스 서버는 잘못된 설계에 의해 발생하는 Overhead를 감당하지 못할 것이다. 그러나 위 구조에서는 모든 회사가 ENTERPRISE라는 테이블에 담겨져 있다. 또한 종업원 또한 EMPLOYEE라는 테이블에 모든 회사의 종업원 정보가 함께 담겨있다. 즉, **위 구조에서는 아무리 많은 데이터가 삽입된다 하더라도 그 데이터는 수직적으로만 쌓이게 된다.** 이러한 테이블 구조는 **중복된 데이터를 생성하는 것으로 인해 발생하는 낭비를 줄이고 필수적인 데이터에 대해서만 메모리를 차지하는 것을 허가한다.** 경영학 분야의 Lean System에서 기본 아이디어를 얻어 설계하였다.

1. **Suitable for Distributed Database**

위 데이터베이스 구조는 최근 각광받고 있는 **“분산형 데이터베이스”로의 변환에 매우 용이**하다. 분산형 데이터베이스 시스템은 너무 많은 데이터를 하나의 저장소에 보관하는 것의 경제성 문제로 인해 등장하였다. 그러나 이러한 분산 시스템을 모든 기성 데이터베이스에 적용할 수 있는 것은 아니다. (할 수는 있겠지만 데이터베이스의 구조를 많은 부분 수정해야한다. 이러한 이유로 기존의 데이터베이스의 정보만을 가져와서 새롭게 분산형 시스템에 맞게 재설계하는 회사도 존재한다. 그러나 이 또한 쉬운 일이 아니다.) 데이터 수에 따라 테이블의 수가 증가하는 구조를 가진 데이터베이스의 경우, 이러한 분산형 시스템을 이용하기에 매우 불리하다. **분산형 데이터베이스 시스템을 적용하기 위해서는 어떠한 “기준”을 가지고 그에 기반하여 각 데이터를 가지는 위치가 달라지게 되는데, 이때 모든 위치에서 가능한 균등한 용량의 데이터를 가지도록 하는 것이 중요하다.**(분산된 여러 개의 저장소 중 한 저장소에 과중이 크게 실리게 되면 분산형 시스템을 적용한 의미를 잃어버리게 된다.) 즉, 위에 언급된 “기준”을 적절히 잡는 것이 분산형 시스템의 핵심이 된다. 다시 돌아가 **현재 설계된 데이터베이스를 보면 이 기준을 적당히 잡아도 데이터의 균등한 분산이 가능**하다. 가령 하나의 회사에 속한 데이터들은 다른 위치의 저장소로 관리하면 탐색에 불리하기 때문에 한 위치에 있어야 한다는 제약이 있다고 하자. 그럼에도 불구하고 위 구조에서 “회사”를 구분해주는 `enterpriseID`가 같은 레코드들은 한 저장소에 관리하면서 그 용량을 회사 단위로 적절히 분산시켜주기만 하면 **당장이라도 분산형 시스템에 적용하는 것이 가능하다.(균등한 용량관리에 유리한 조건으로)**

1. **Lean & Lean**

현재 데이터베이스의 스키마는 낭비를 최소화하기 위해 최대한 데이터를 중복으로 관리하거나 불필요한 데이터를 가지는 것을 지양한다. 이러한 Lean 지향의 한 예로, **위 테이블에는 직원의 월급이라는 SALARY 테이블은 존재하지 않는다. 그러나 제공하는 서비스에서는 종업원은 자신의 급여를 확인할 수 있고, 회사 또한 자신의 회사에 속한 종업원들의 급여를 한 눈에 확인할 수 있다.** 서비스에 구현된 쿼리문은 구성된 테이블에서 적절한 조건에 해당하는 값들을 가지고 계산식을 기반으로 그에 대한 정보를 제공한다. **각 회사, 개인은 자신과 관계없는 데이터에 접근하는 것을 방지하면서 테이블 공간은 공유해야하기 때문에 쿼리문을 적절히 작성하는 것이 매우 중요하다.** 그 부분에 대해서는 기나긴 쿼리문으로 구현된 상태이다.