문제 1.
처음에는 역치라고 하셔서 설마 했는데 설마를 넘어 매우 세상 힘들었습니다. 심지어 제가 비전공자라서 용어에 관련해서 아는 단어도 몇 개 없었으며, 다른 분들은 전부 아는 듯이 용어를 말하며 넘어가길래 제가 수강을 잘못했나? 라는 심정으로 포기해야되는지 고민을 많이 했습니다. 하지만 강사님께서 코딩에는 답이 없고 시도를 계속 하면 답이 나온다는 말과 강사님께서 운영하는 Notion을 보며 배울게 많구나 라는 심정으로 울며 겨자먹기식으로 멱살 잡고 따라가려고 노력하고 있습니다. 물론 나중에 알게 된 사실이지만 저나 다른분들이나 몇 분을 제외하고 저랑 같다는건 나중에 알았지만요.
결론은 앞으로도 계속 어려울거라고 생각은 하지만 저는 계속 배워갈 것이고 배우는 속도가 느리더라도 따라 갈 수 있도록 포기하지 않고 노력하겠습니다.

문제2.
분석 작업과 설계 그리고 구현까지 통일된 방식으로 커뮤니케이션 가능하며, 일반적으로 많이 사용되는 데이터 중심의 접근이 아닌 순수한 도메인 모델과 로직에 집중하여 구축이 용이하다. 또한 협업 기준으로 내가 어떻게 작업을 하는지 어떤 방향으로 만들고 있는지 이해 하기 용이하며 빠르게 편입이 가능하다.

문제3.
개발을 진행하다 꼬이더라고 테스트를 돌려봄으로써 안심하고 진행이 가능하며, 개발 중간중간 테스트를 활용하여 문제가 되는 구간을 찾을 수 있으며,이로 인해 재설계의 시간을 단축시킬 수 있다.

문제4.
TDD는 DDD에서 만들어진 기능을 테스트하여, 해당 기능이 동작하는지 확인할 수 있습니다. 이와 같이 DDD 주도로 설계될 때 새로 추가되거나 변경되는 부분에 한해 TDD를 활용하여 오류 및 문제점을 적발 하여 좀더 신뢰성 있는 코드 개발이 가능합니다.

문제5.
사람들 마다 각자의 능력과 스킬이 다르기 때문에 개개인의 능력보다는 팀원간의 소통이 제일 중요하다고 생각합니다. 팀원 모두가 잘한다면은 사람만 보면 정말 좋은 결과물을 내놓을 수 있지만, 소통이 없다면 그 결과물은 어떻게 나올지 장담할 수 없습니다. 그렇기에 팀에 나사 빠진 사람이 몇 명 있더라도 소통을 통하여 서로 문제되는 부분을 보완하고 완벽하진 않더라고 좋은 성과를 내는 것이 협업이라고 생각합니다.

문제6.
함수 포인터를 활용하는 이유는 코드 개발중 새로운 기능이 추가될 때, 해당 기능을 제공하는 새로운 함수를 추가하고, 인터페이스를 구현하는 함수 포인터를 제공함으로서 시스템 확장에 용이합니다 또한 함수간의 의존성을 해결하며, 함수 포인터를 사용하면 런타임 시 의존성을 동적으로 해결이 가능합니다. 또한 변경 및 오류가 발생했을 때 포인터를 사용하면 해당 함수만 수정하면 되기에 개발의 유지보수 및 편의성을 높여줍니다.

문제7.
virtual method를 사용하면 기본 클래스에서 정의된 메서드를 하위 클래스에서 오버라이딩 할 수 있으며, 상속을 통해 코드를 재사용하고, 새로운 기능을 추가하거나 기존 기능을 수정할 때 유연성을 제공하며, 인터페이스나 추상 클래스에서 virtual method를 정의하면, 이를 구현하는 각 클래스에서 해당 메서드를 반드시 구현해야해서 코드의 일관성을 유지시켜줍니다. 또한 새로운 클래스를 추가하거나 기존 클래스를 수정하지 않고도 새로운 동작의 도입이 가능합니다.
그리고 가상 소멸자를 사용하면 기본 클래스의 포인터를 통해 파생 클래스의 인스턴스를 삭제할 때 올바른 소멸자가 호출되며, 이것은 메모리 누수를 방지합니다.

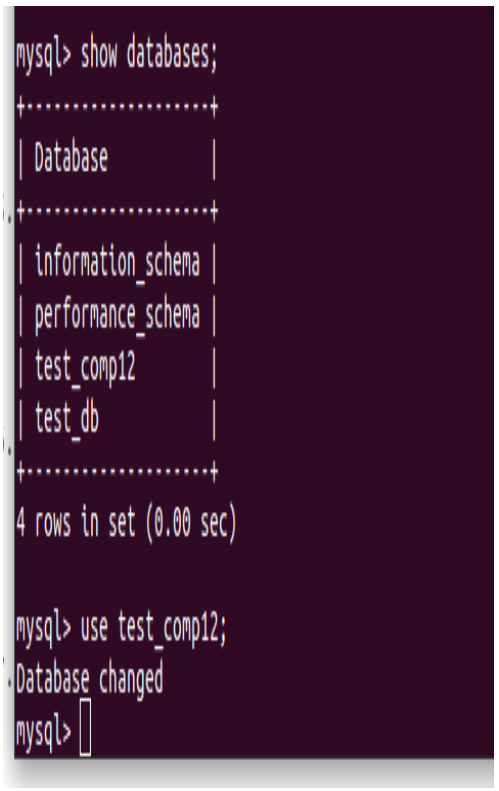
문제 8.
어려운 문제가 생긴다면 나 혼자만의 문제로 진행을 못하게 되어 내가 굴린 스노우볼로 협력에 차질이 생길 수가 있기 때문에 팀원들에게 현재 상황을 공유하며, 진행을 잠시 늦추어 해당 문제를 같이 풀어나가는게 가장 좋을 방법이라고 생각합니다.

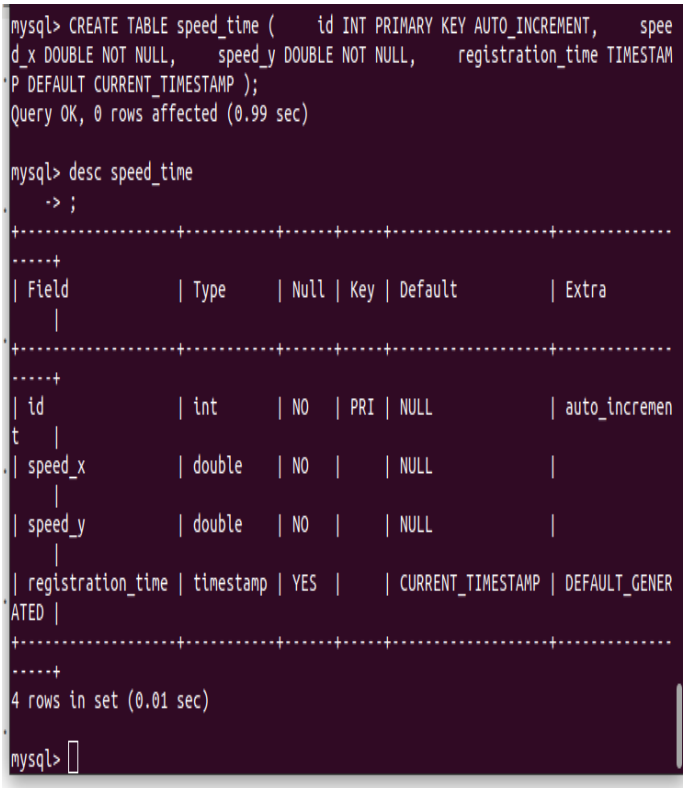
문제 9.
Backlog를 작성하게 되면 프로젝트의 목표 및 요구사항을 시각화가 가능합니다. 이를 통해 어떤 작업을 먼저 수행해야하는지 결정할 수 있으며, 중요한 작업에 우선순위를 부여가 가능합니다 또한 프로젝트에 대한 이해를 공유하며 팀간의 협업을 강화합니다. Backlog를 통해서 팀원들이 어떤 작업을 진행하는지 확인이 가능하며, 중복된 작업을 피할 수 있으며,이를통해 어떤 미래를 다가올지 미리 두려워 할 수 있습니다. 그리고 Backlog를 작성하면서 추가적으로 어던 작업이 필요한지 파악이 가능하며, 새로운 기능이나 개선 사항을 추가하고, 지속적인 개선이 가능합니다.

문제 10.
역량의 부족과 이해도가 떨어져서라고 생각합니다. 팀원과의 소통을 통해 방향성을 잡는다고 해도 그 코드를 짜는 것은 제가 해야하기에 기능만 생각해서 구현하다 보니 좀 더 단순하고 가독성이 좋은 방법이 있었겠지만 이것 저것 넣다보니 기능 구현은 되었지만 만들고 보니 젤나가도 이마를 탁 치게 만드는 그러한 걸작이 탄생하게 되었습니다.
(사설이지만 애초에 혼종이 만드니 혼종이 탄생하는 거 같습니다.)

문제 11.
기능단위의 Backlog가 작성된다면 내가 지금 진행하고자 하는 방향성을 잃고 목적이 불분명해지게 됩니다. Backlog라는것은 프로젝트의 목표 및 요구사항을 시각화지만 기능 단위로 작성하게 되면 내가 어떤 방향을 가지고 어떤 목표를 가지고 작성했는지 모호해지며, 갈수록 목적이 퇴색되기 때문에 용두사미 와 같은 결과가 나올 수 밖에 없습니다.

[복합 문제 12 ~ 20]
13.





문제 21.
버스정류장 같은 특정한 장소는 계속해서 사람이 오고 가고 하는 장소이기에 계속해서 스토리지가 쌓일 수 밖에 없습니다. 그러므로 현재 사람들이 붐비는 구간을 찾고자 하는 목표가 있기에 분석결과가 중요한 시간대의 결과값을 저장하고, 필요한 경우에만 영상 데이터를 조회시켜 결과를 다시 계산하는 방법을 도입하고, 특정 시간대 주로 한산하여 사람이 적은 구간의 데이터를 주기적으로 아카이빙 하거나 삭제하는 정책을 수립하여 스토리지 공간을 확보할 수 있습니다.

문제 22.
포인터는 프로그래밍 언어에서 메모리를 효과적으로 활용하고, 동적으로 데이터를 할당하고 조작하기 위해 사용됩니다. 또한 포인터를 사용하면 메모리의 주소에 직접 접근하여 데이터를 저장하거나 검색이 가능하며, 프로그램 실행중에 동적으로 메모리를 할당하고 해제가 가능합니다.

문제 23.
함수들이 서로 스택을 공유하지 않는 이유는 주로 함수 호출과 반환의 관리, 데이터의 독립성, 보안등 다양한 이유로 발생하며, 각 함수는 자체적으로 호출 스택을 유지하고,

함수가 호출될 때마다 새로운 프레임이 스택에 추가되며 이 프레임은 함수의 지역변수,매개변수,복귀 주소등을 저장하는데 사용됩니다. 또한 각 쓰레드는 독립적인 호출 스택을 가질 수 있으며 이로 인해 각 쓰레드 간에 함수 호출과 관련된 데이터가 충돌하지 않도록 합니다.

문제 33.
해당 Entity의 역할과 책임을 다시 검토하고, 특히 복잡한 작업을 수행하는 부분을 별도의 도메인 클래스나 서비스로 분리하거나, 작업의 복장성이 증가하면 새로운 도메인을 식별하여, 해당 도메인을 적절한 클래스로 분리하고 TDD를 활용하여 테스트 케이스를 작성하여 코드의 안전성을 확인한다.

죄송합니다…