# 최종 결과 보고서

## 고객 및 채팅 관리 프로그램

오스템임플란트

C++ 기반 영상처리 전문가 양성과정

안다미로

# CONTENTS

## 개요

- 프로젝트 개요
- 개발 환경
- 프로그램 요약

## 프로젝트 내용

- 1차 프로젝트
  - 고객 관리 프로그램 설계
- 2차 프로젝트
  - Qt를 이용한 GUI 전환
  - 채팅 프로그램 설계
- 3차 프로젝트
  - ORACLE을 이용한 DB설계

## 결론

- 보완 사항
- 발전 방향

# 1

# 개요

1-1. 프로젝트 개요

1-2. 개발 환경

1-3. 프로그램 요약

# 1-1. 프로젝트 개요

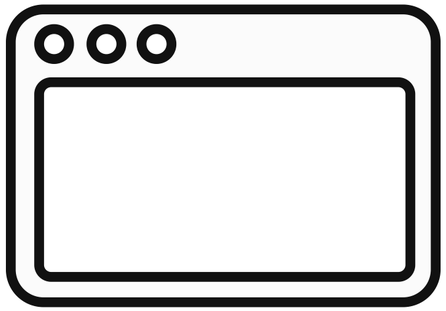
| | | |
|---|---|---|
| **프로젝트** | 고객 관리 및 채팅 프로그램의 제작 | |
| **목적** | 고객, 상품, 주문 정보를 관리하고, 고객과 채팅할 수 있는 프로그램을 제작 | |
| **개발 기간** | 1차 | 2022년 09월 01일 ~ 2022년 09월 07일 (7일) |
| | 2차 | 2022년 10월 17일 ~ 2022년 10월 30일 (14일) |
| | 3차 | 2022년 11월 10일 ~ 2022년 11월 20일 (11일) |
| **개발자** | 안다미로 | |
| **버전 관리** | 1차 | https://github.com/AndaCondaa/Project_CustomerManagement_CPP |
| | 2차 | https://github.com/AndaCondaa/Project_CustomerManagement_Qt |
| | 3차 | https://github.com/AndaCondaa/Project_CustomerManagementApp |

# 1-2. 개발 환경

| | |
|---|---|
| OS | Windows 10 |
| Language | C++ |
| GUI Framework | Qt 6.3.1<br>Qt 6.3.2(Source for DB) |
| IDE | Qt Creator 8.0.1 |
| DB | ORACLE XE 11g |
| Version | Git |




# 1-3. 프로그램 요약

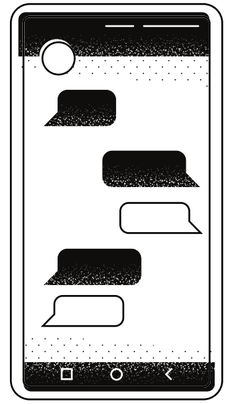


## <관리 프로그램>

- 고객, 상품, 주문 관리
- 채팅 서버
- 채팅 로그 저장

## <관리자용 채팅 프로그램>

- 서버를 통해 고객과 채팅
- 파일 전송
- 공지사항 등록



## <고객용 채팅 프로그램>

- 서버를 통해 관리자와 채팅
- 파일 전송

# 2

# 프로젝트 내용

## 2-1. 1차 프로젝트 요약

- 전체 프로그램 설계
- 책임 중심 설계 기법 적용
- 파일저장 포맷 설계

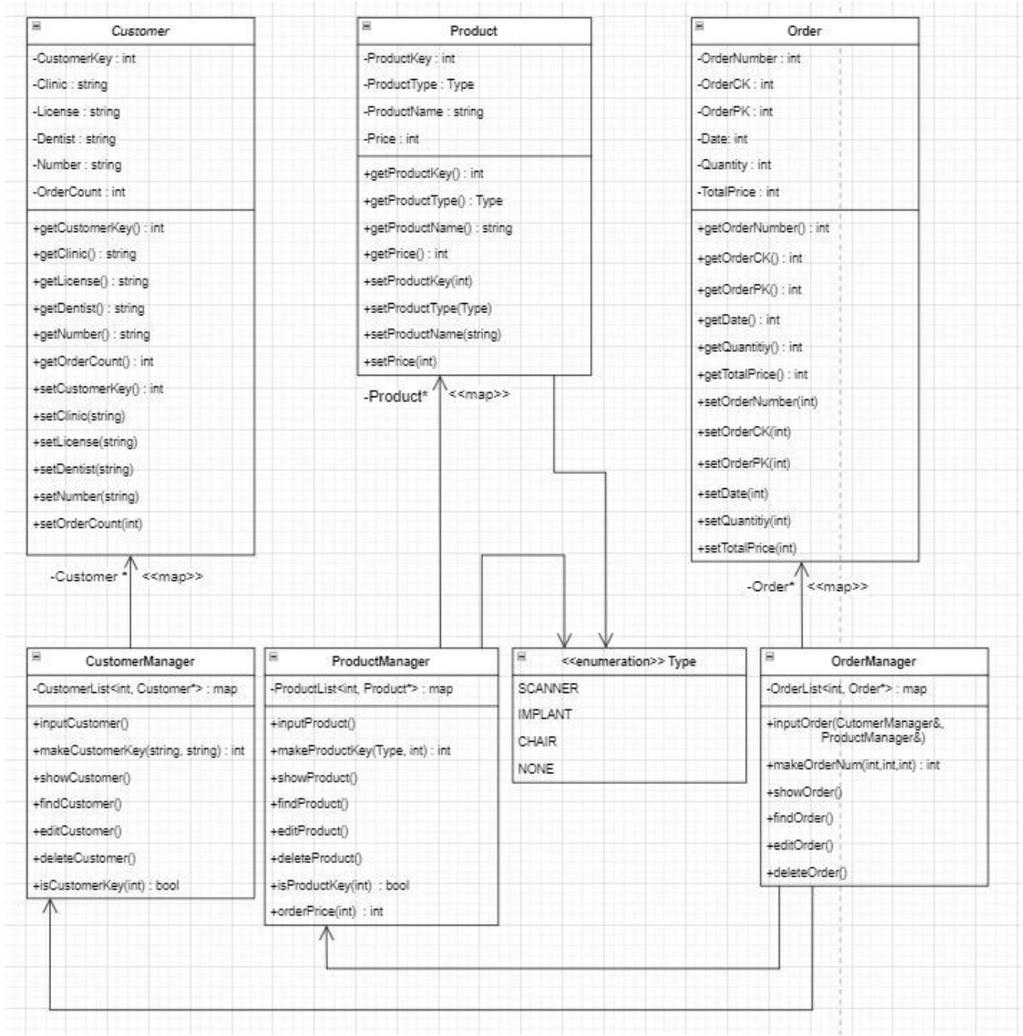
## 2-2. 2차 프로젝트 요약

- GUI 전환
- 채팅 구조 설계
- 채팅 프로토콜 설계
- 첨부 파일 전송
- 멀티스레드를 이용한 로그 저장

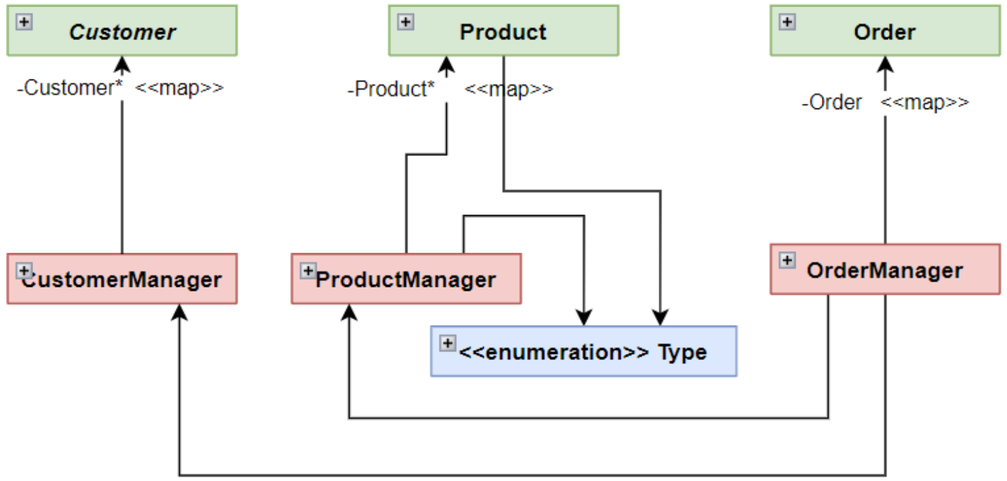
## 2-3. 3차 프로젝트

- ORACLE DB
- DB 설계_데이터 관계
- DB 설계_권한 설정
- DB에 UI 적용_MVC패턴
- 로그 저장_생산자-소비자 패턴

# 2-1. 1차 프로젝트 요약
## - 전체 프로그램 설계




<UML>




- 고객, 상품, 주문클래스에 각각 관리클래스를 추가한 형태로 설계
- 고객, 상품, 주문클래스에서 하나의 객체는 하나의 고객정보, 상품정보, 주문정보를 의미
- 상품 타입을 정형화하는 enum 클래스를 추가
- 주문 관리클래스는 고객과 상품의 정보가 있어야 기능을 구현할 수 있으므로, 고객관리 클래스와 상품관리 클래스를 참조

# 2-1. 1차 프로젝트 요약
## - 책임 중심의 설계 기법 적용

**목적**

- 데이터 중심 설계에서 책임 중심 설계로 변경
- 객체지향 프로그래밍의 핵심은 역할과 책임

**기준**

- 캡슐화
- 응집도와 결합도를 고려한 설계 구현

**방법**

- 변경될 가능성이 높은 부분은 클래스 외부로 노출되지 않도록 설계
- 접근지정자와 getter, setter 함수 등을 이용하여, 유연하게 관리

# 2-1. 1차 프로젝트 요약
## - 책임 중심의 설계 기법 적용

캡슐화 → 보완

- 객체 단위에서 변경 가능성이 높거나, 보안을 위해 내부로 숨겨할 데이터는 Private으로 설정
- getter, setter 함수는 protected 또는 경우에 따라 public으로 유연하게 변경

- 객체 외부에서 내부로 접근할 때, 개념적으로 정확한 구현을 위해 Interface 필요
- 인터페이스 구현을 통해 외부에서 부모 클래스를 통해 호출할 수 있지만, 객체에 직접 접근할 수는 없도록 제한

# 2-1. 1차 프로젝트 요약
## - 파일 저장 포맷 설계

**목적**

- 파일 저장을 통해 연속된 프로그램 사용 구현

**방법**

- C++ ifstream, ofstream을 이용하여 파일 입출력 구현
- 객체별로 정보의 종류를 묶어야 하므로, CSV형태로 저장

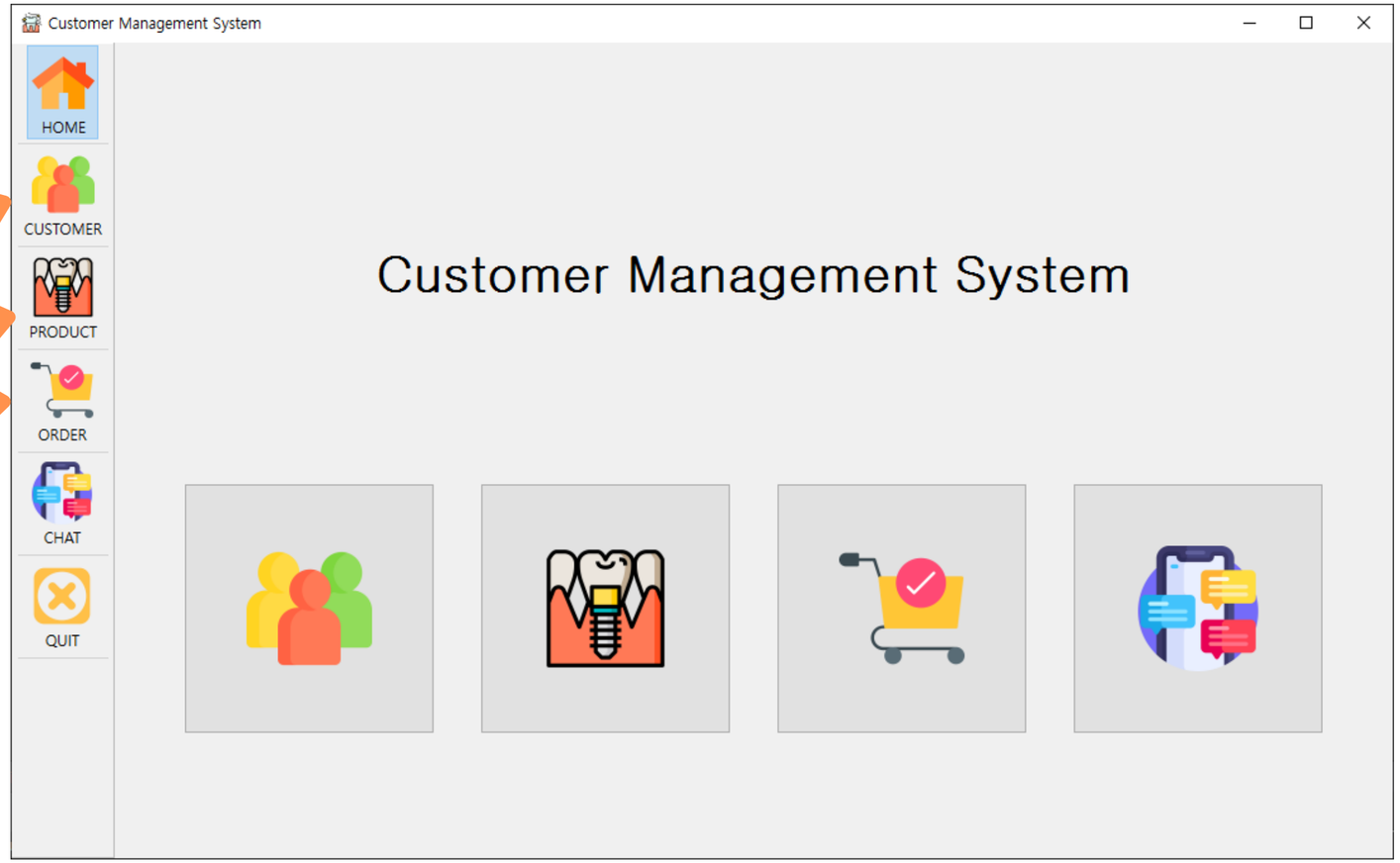
**구현(예시)**

8245, 참조은치과병원, 98-1234-12, 참조은원장, 0212341234, 1
1311, 박교정전문병원, 10-4567-89, 교정전문의, 01012341234, 2
5555, 오스템치과, 87-1234-43, 미스오스템, 02211112222, 1
8245, 새로운치과병원, 19-1234-12, 박새로운, 0212341233, 0

# 2-2. 2차 프로젝트 요약
## - GUI 전환

**메인 페이지**

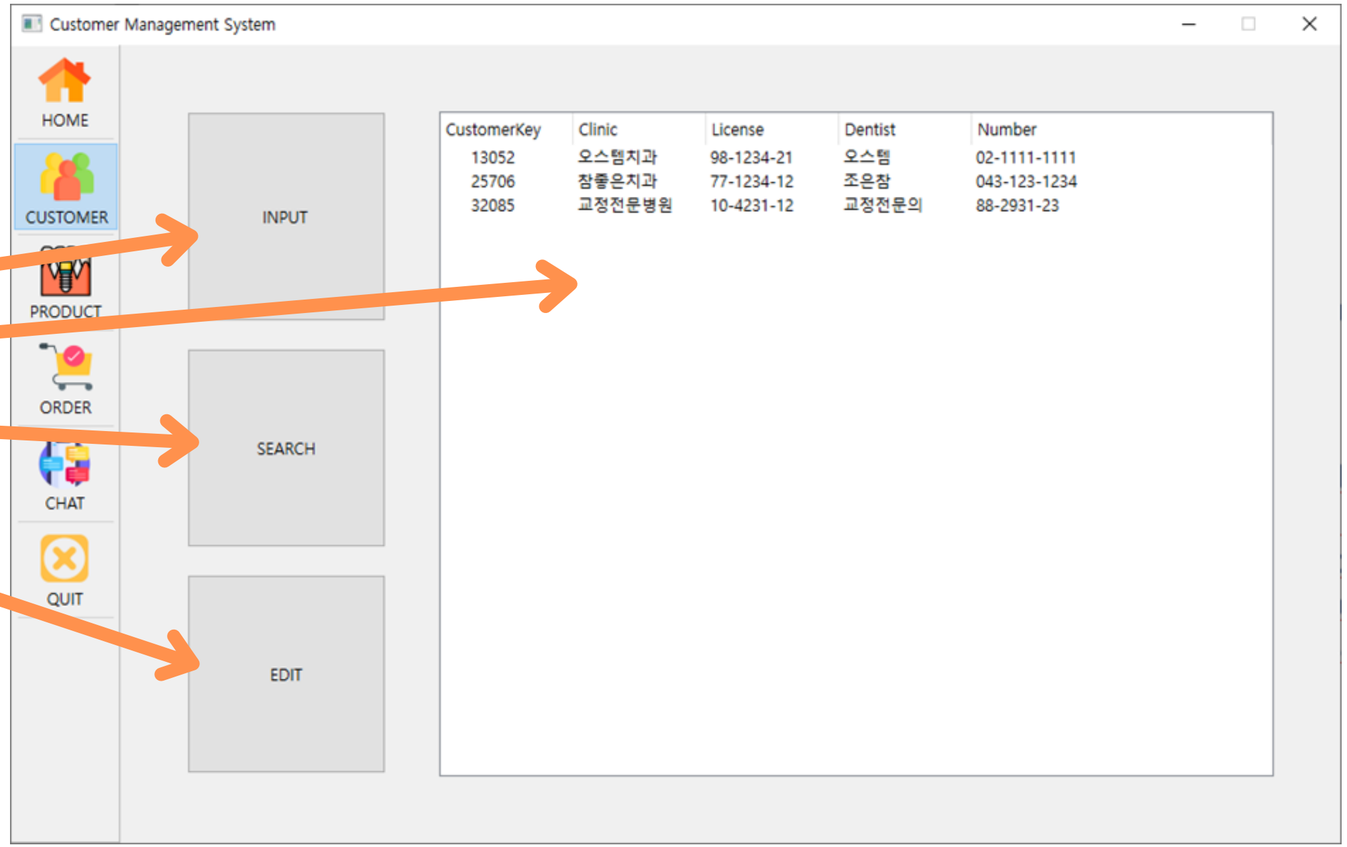
```
******프로그램 메인******

1 - 고객 프로그램
2 - 상품 프로그램
3 - 주문 프로그램
4 - 종료

입력:
```



# 2-2. 2차 프로젝트 요약
## - GUI 전환

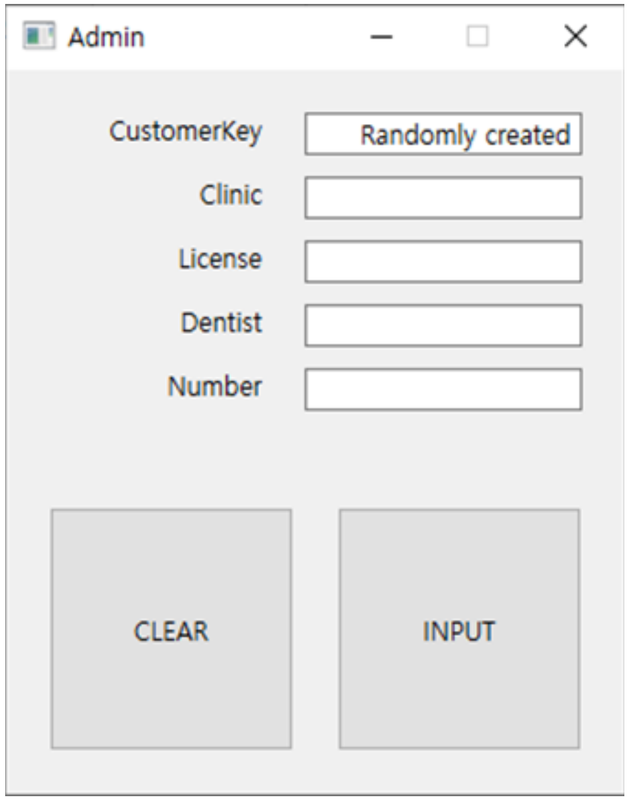
**관리클래스 메인**

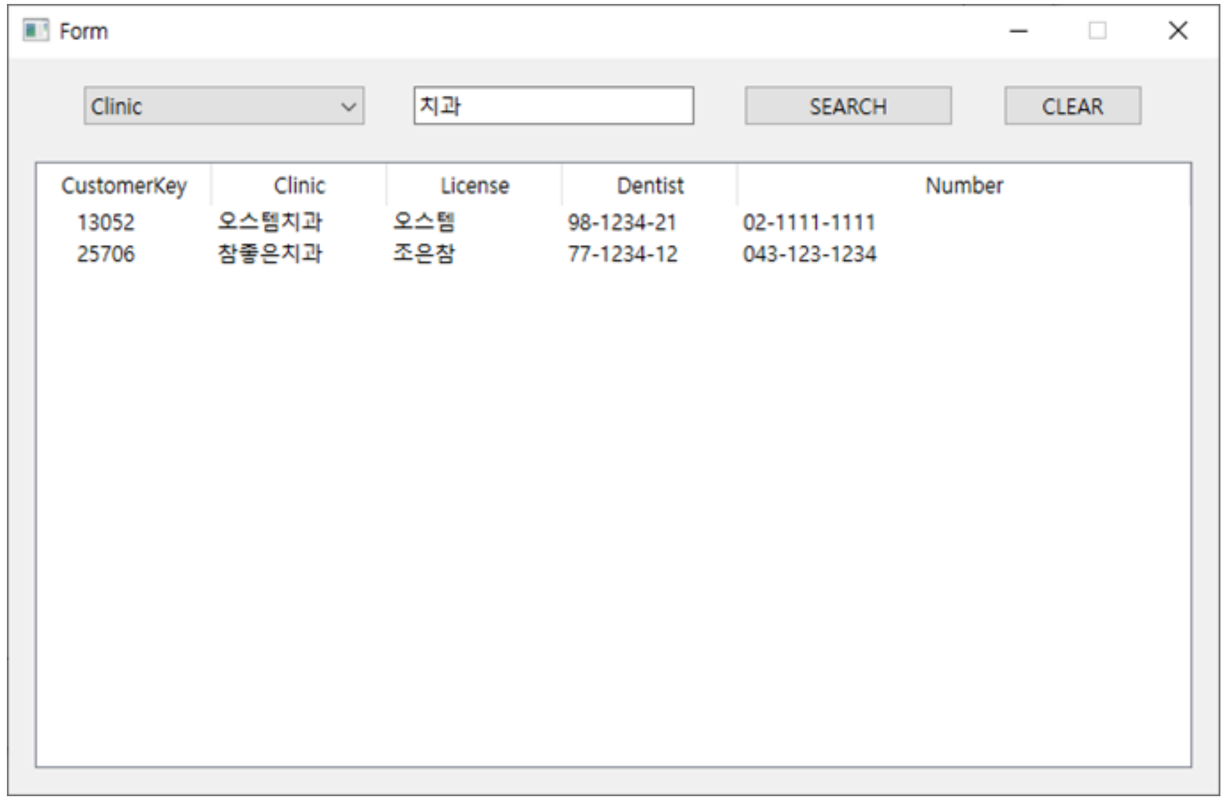
```
고객관리 프로그램

1. 신규고객 추가
2. 고객목록 출력
3. 고객 검색
4. 고객정보 수정
5. 고객 삭제
6. 메인으로 돌아가기

입력:
```

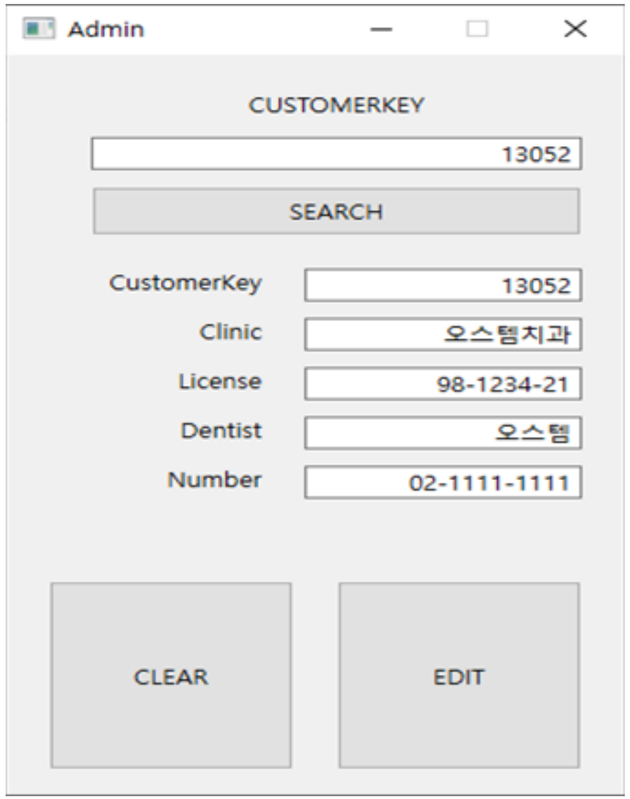


# 2-2. 2차 프로젝트 요약
## - GUI 전환

**기능 클래스**



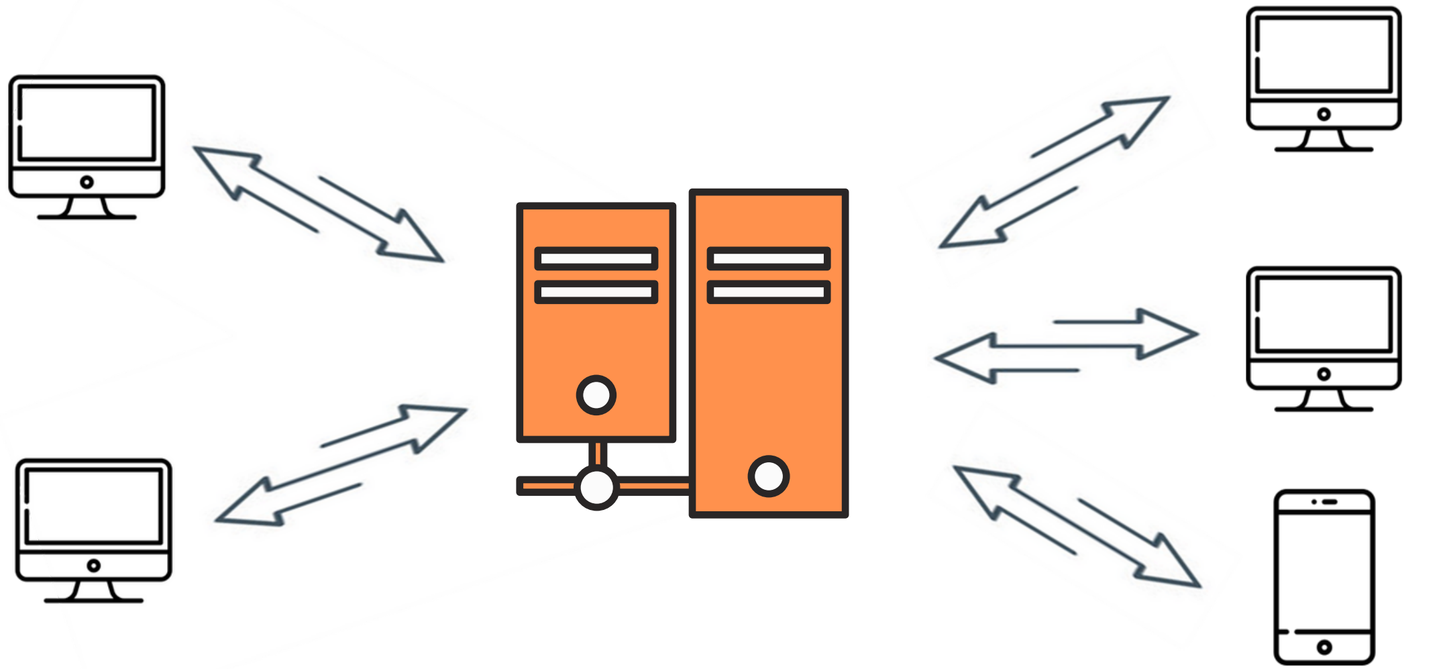




<Input Class>

<Search Class>

<Edit Class>

# 2-2. 2차 프로젝트 요약
## - 채팅 구조 설계

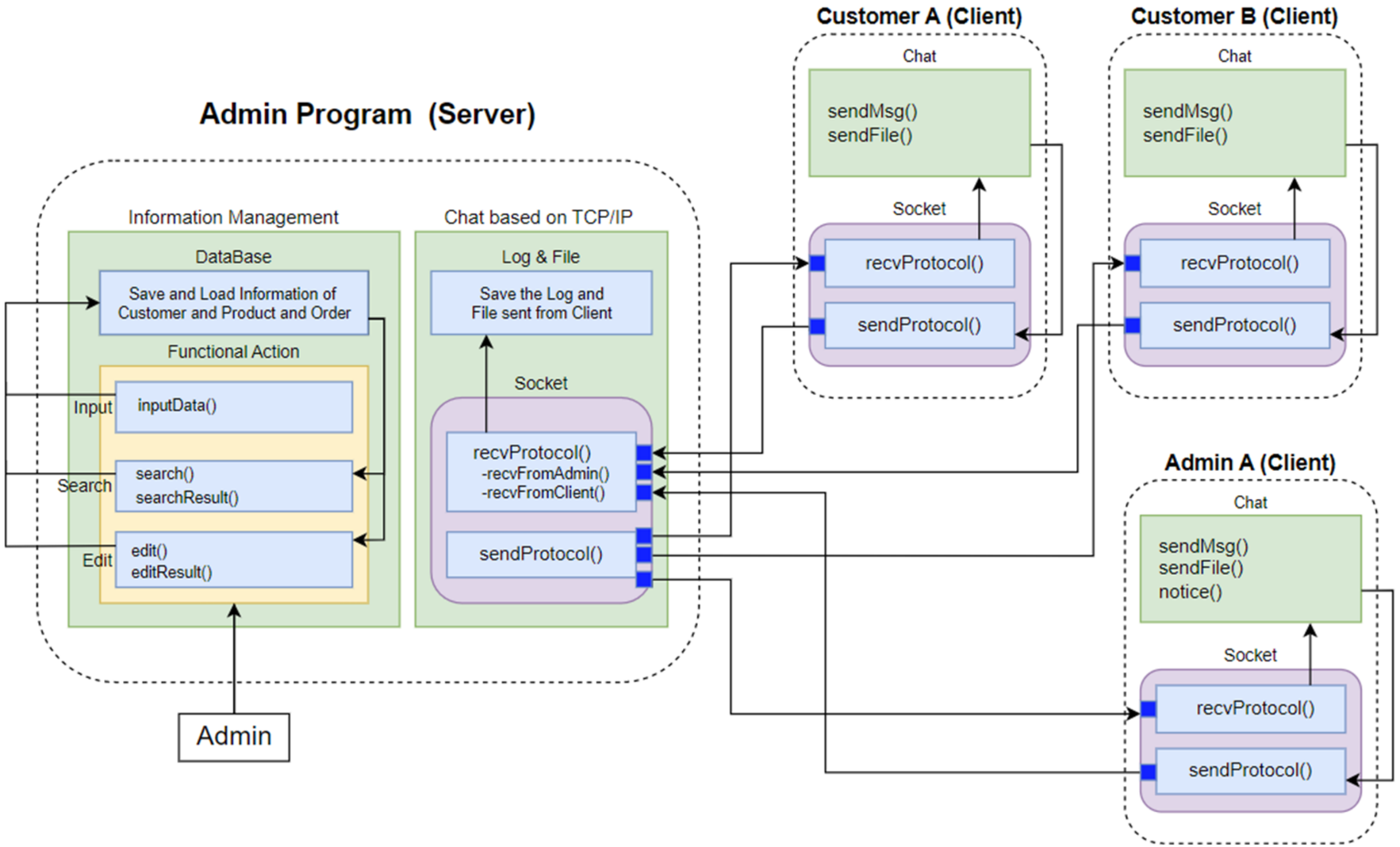


관리자 서버 고객

1:N 채팅

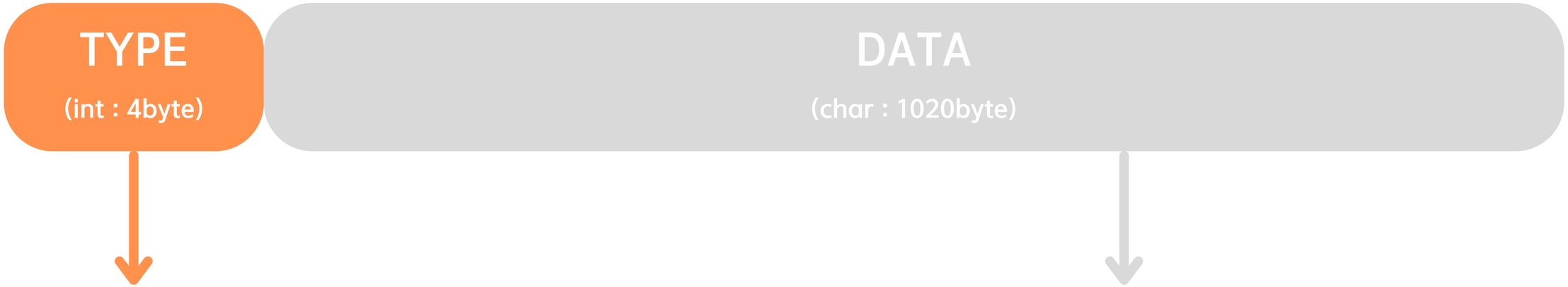
- TCP/IP 통신으로 구현
- 관리자 기준 일대다 채팅 구현
- 기존 관리자 프로그램이 서버 역할
- 두 클라이언트(관리자, 고객)가 서버를 사이에 두고 채팅하도록 설계

# 2-2. 2차 프로젝트 요약
## - 채팅 구조 설계




Functional Block Diagram

# 2-2. 2차 프로젝트 요약
## - 채팅 프로토콜 설계



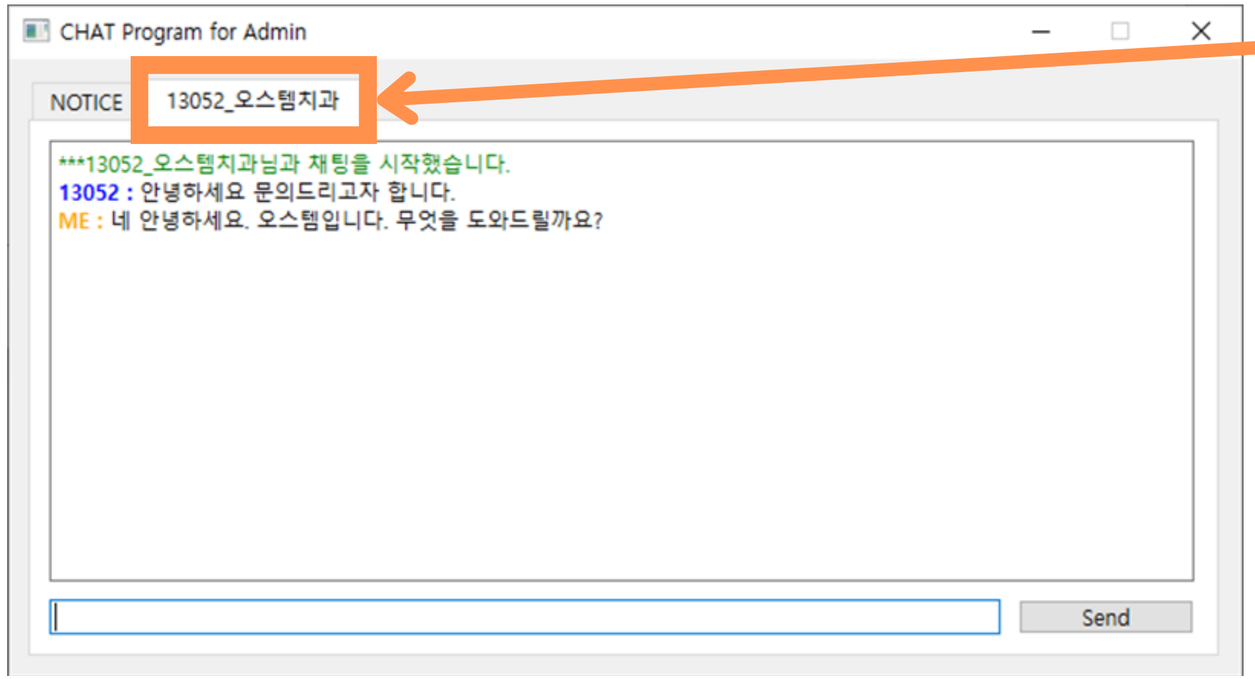

| | |
|---|---|
| Sign_In | 고객 로그인 |
| Sign_In_Fail | 고객 로그인 실패 |
| Admin_In | 관리자 채팅 로그인 |
| Admin_In_Fail | 관리자 채팅 로그인 실패 |
| In | 고객 채팅방 접속 |
| In_Fail | 고객 채팅방 접속 실패 |
| Out | 고객 채팅 종료 |
| Message | 채팅 메시지 전송 |
| Invite | 채팅 초대 |
| Kick_Out | 채팅에서 강제퇴장 |
| Close | CloseEvent 발생 |
| Notice | 공지사항 전송 |

<DATA>

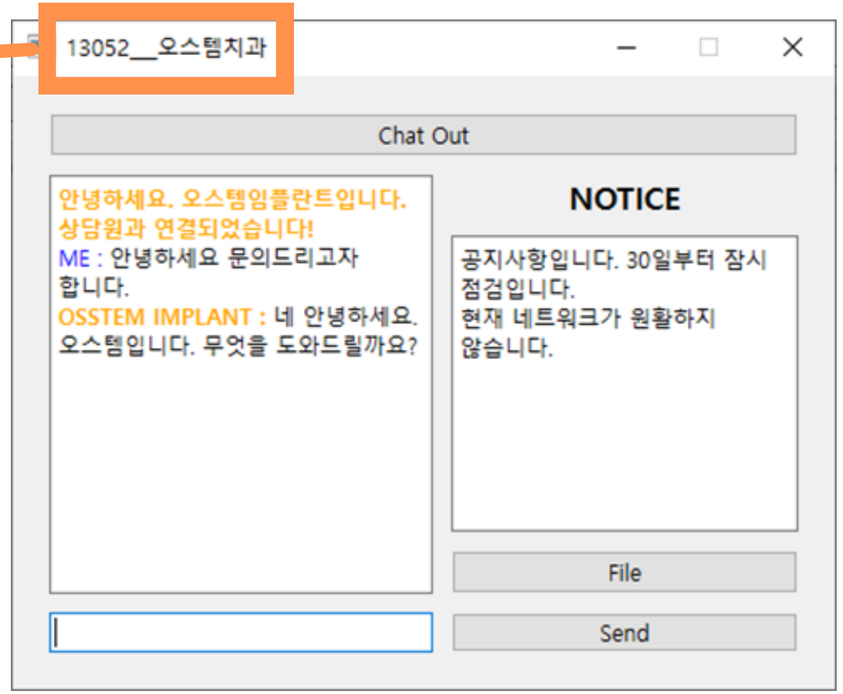
- 기본적으로 Qstring Data 전송
- 필요한 경우 구분자를 사용
- " | " : 짝 형태의 데이터 구분 (CustomerKey|Name)
- " || " : 여러 종류의 데이터를 보낼 때의 구분

# 2-2. 2차 프로젝트 요약
## - 채팅 프로토콜 설계

**관리자용 채팅 프로그램**

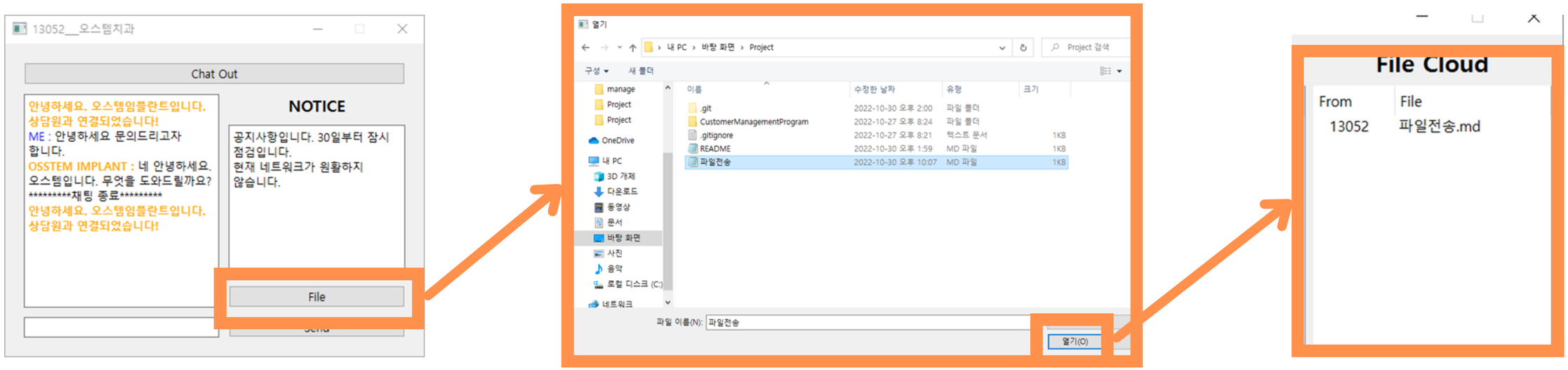



**고객용 채팅 프로그램**




- 고객이 채팅방에 입장하면, 관리자 프로그램에는 탭이 추가되는 형태로 채팅방 생성
- 고객은 오직 한 명의 관리자와 채팅 가능
- 관리자는 탭이 추가되는 형태로 여러 명의 고객과 채팅 가능

# 2-2. 2차 프로젝트 요약
## - 첨부 파일 전송

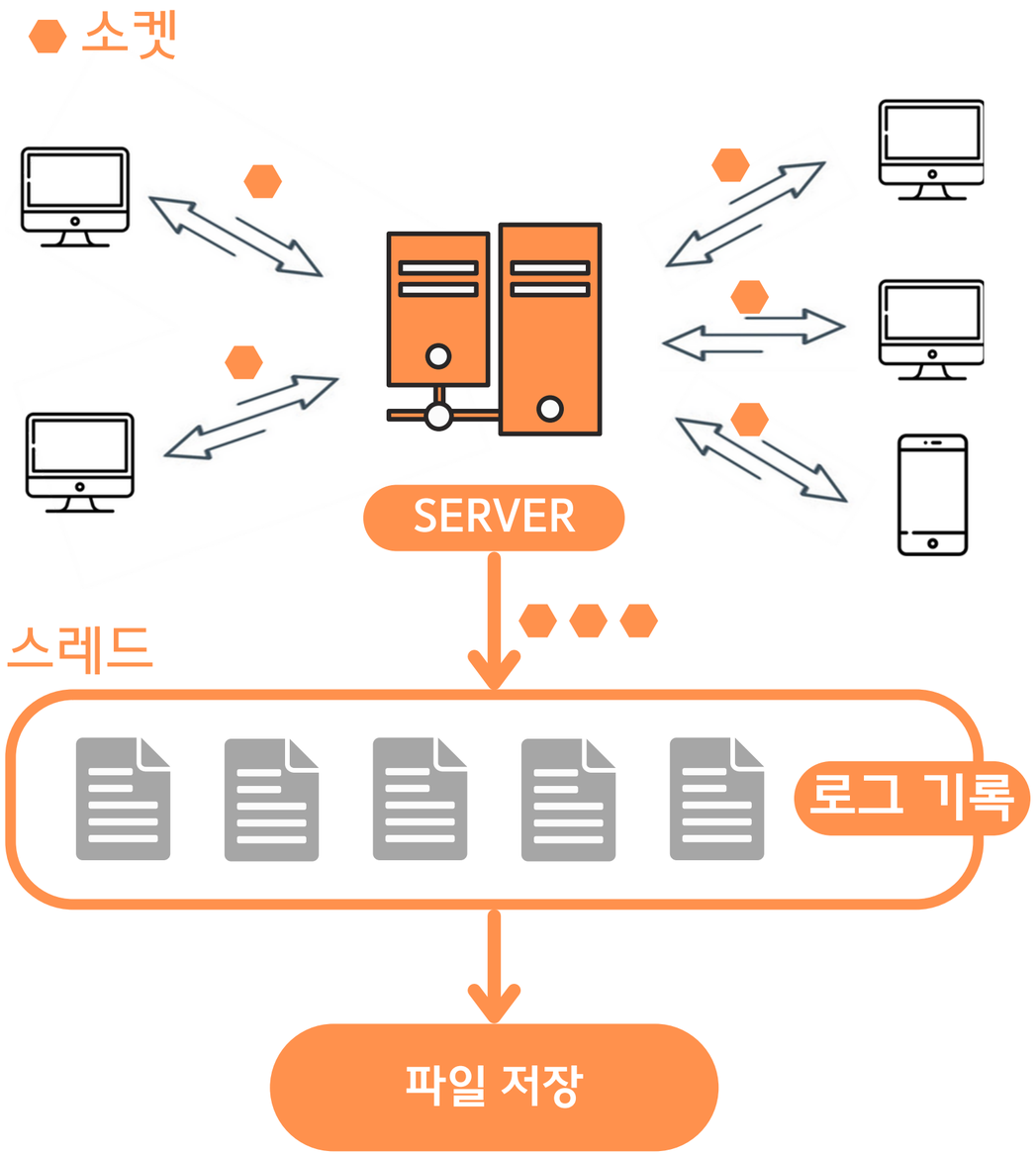



- 파일 버튼을 누르면 파일 다이얼로그 생성
- 전송할 파일을 누르면 파일 전송
- 채팅 매니저 내에 파일클라우드 목록에 저장되고, 해당 파일은 지정된 경로에 저장

# 2-2. 2차 프로젝트 요약
## - 첨부 파일 전송


- 파일 전송의 경우, 정해진 블럭 단위로 파일을 쪼개어 여러 개의 패킷으로 나누어 전송
- 첫 번째 패킷에는, 총 용량과 파일 이름, 클라이언트의 정보를 포함
- 서버에서는 패킷을 받을 때마다, 총 용량과 비교
- 받은 패킷의 합산 용량이 총 용량과 같으면, 모든 패킷이 전송되었다고 판단

# 2-2. 2차 프로젝트 요약
## - 멀티스레드를 이용한 로그 저장




- TCP/IP 방식으로 통신하면서, 서버에서는 클라이언트에서 보낸 소켓을 읽어서 기록
- IP, PORT, DATE, MESSAGE 정보를 기록
- 서버에서 스레드가 동작하면서 주기적으로 기록된 로그를 파일로 저장

# 2-3. 3차 프로젝트
## - ORACLE DB

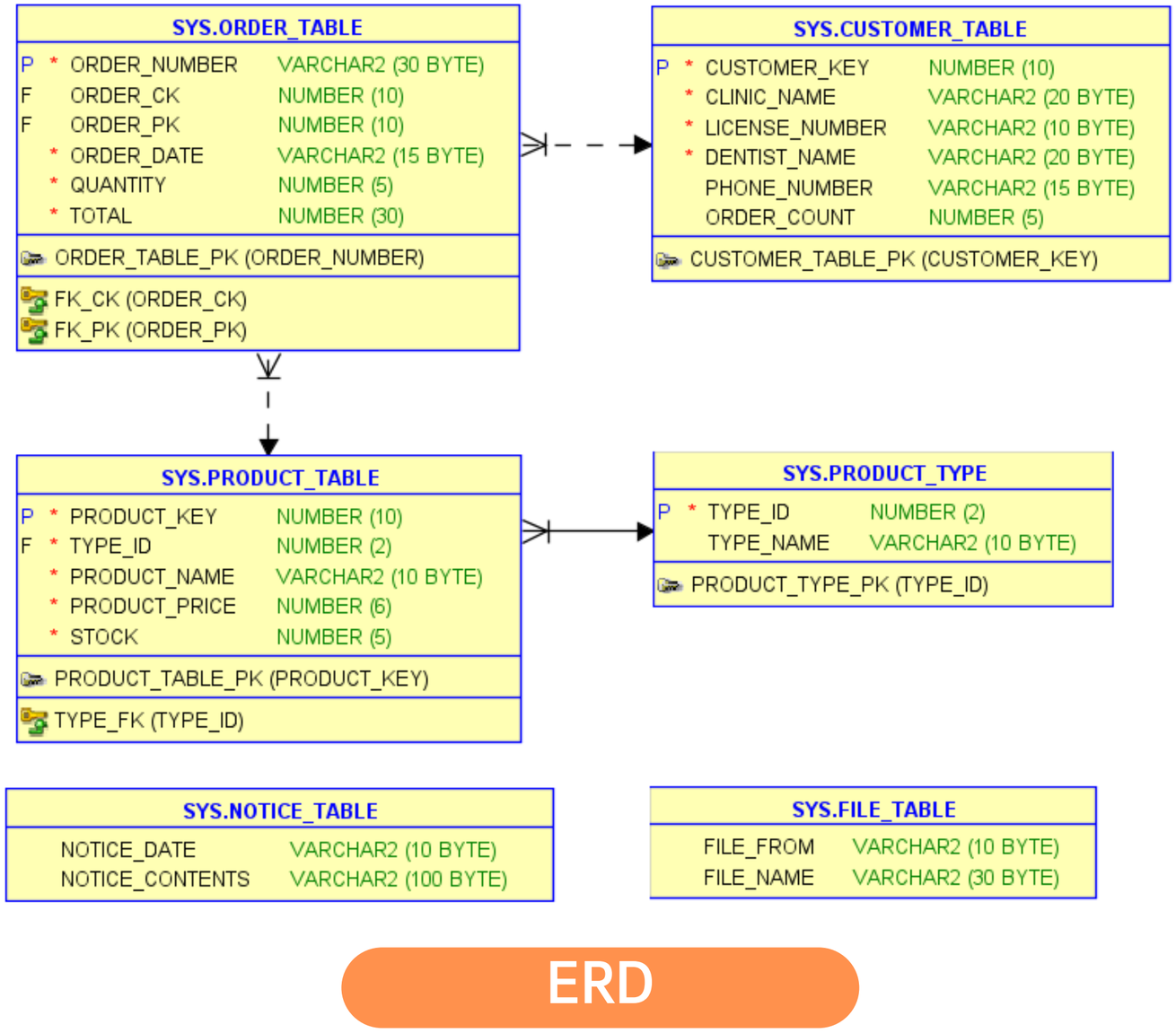
**상황**

- 프로그램 구조 상, 다수의 관리 매니저들이 하나의 데이터 베이스에 접근할 수 있음
- 등록되는 고객, 상품, 주문이 많아지면, 유기적으로 연결된 대규모 데이터베이스를 처리해야 하는 수요가 발생할 수 있음
- 임상데이터 등 사내에서 발생하는 데이터는 개인적으로 민감한 정보인 경우가 대다수
- 보안 관리가 철저하게 이루어져야 함

ORACLE DATABASE

- 다수의 사용자 접근 가능
- 타 데이터 베이스보다 고성능 트랜잭션 수행
- 대용량 데이터베이스 지원
- 분산처리 시스템 지원
- 높은 보안성

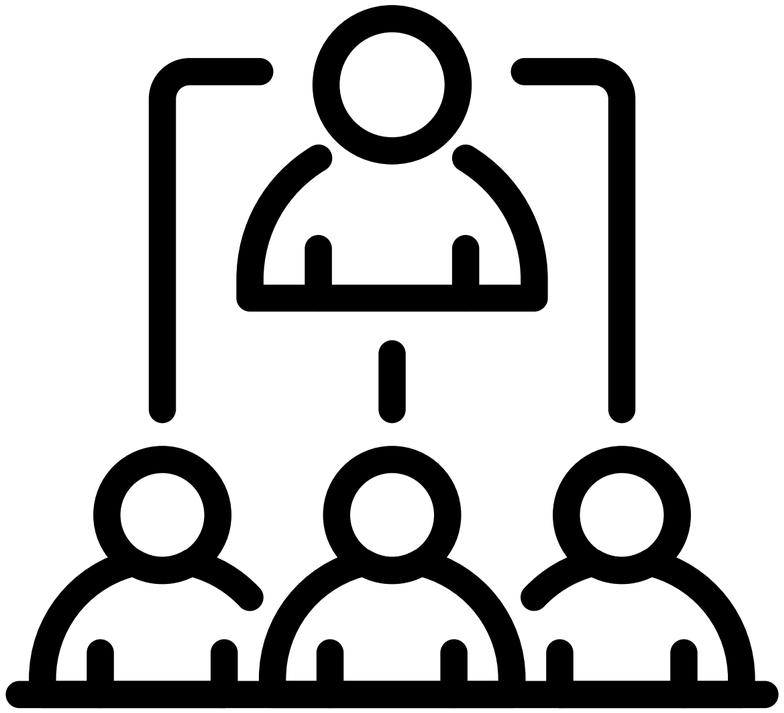
# 2-3. 3차 프로젝트
## - DB 설계_데이터 관계




ERD

- 기존 CSV형식의 파일로 저장하던 정보를 DB로 전환
- 필요한 데이터를 RDBMS에 맞게 연관된 테이블로 설계
- 테이블마다 PK설정 및 필요한 경우에 FK를 적절하게 설정하여, 유기적인 데이터 베이스 구조 설계

# 2-3. 3차 프로젝트
## - DB 설계_권한 설정



**DBA**

- DataBase를 직접 관리하는 것은 DBA
- 테이블 생성, 삭제, 데이터 입력, 삭제, 조회 등 모든 권한을 가지고 있음
- 이후 다른 매니저가 업무상 특정 테이블에 대한 권한이 필요한 경우, 계정 생성 및 권한 부여 가능

**MANAGERS**

- DBA로부터 계정과 권한을 부여 받아서, 권한에 해당하는 만큼 DB에 접근할 수 있음
- 기본적으로 특정 테이블에 대한 SELECT 권한만 부여
- 데이터를 추가하는 권한이 필요한 경우, INSERT에 대한 권한을 주는 것은 지양하고, 프로시저나 함수를 생성하여 권한을 부여

# 2-3. 3차 프로젝트

## - DB 설계_권한 설정

```
GRANT SELECT ON customer_table TO customer_manager;
GRANT SELECT ON product_table TO product_manager;
GRANT SELECT ON product_type TO product_manager;
GRANT SELECT ON order_table TO order_manager;
GRANT SELECT ON notice_table TO chat_manager;
GRANT SELECT,INSERT ON notice_table TO chat_admin;
GRANT SELECT ON notice_table TO client;
GRANT SELECT ON customer_table TO chat_manager;
GRANT SELECT ON customer_table TO chat_admin;
GRANT SELECT, INSERT ON file_table TO chat_manager;
```

테이블 접근 권한 부여

```
GRANT EXECUTE ON INPUT_CUSTOMER TO customer_manager;
GRANT EXECUTE ON INPUT_PRODUCT TO product_manager;
GRANT EXECUTE ON INPUT_ORDER TO order_manager;
GRANT EXECUTE ON EDIT_CUSTOMER TO customer_manager;
GRANT EXECUTE ON EDIT_PRODUCT TO product_manager;
GRANT EXECUTE ON EDIT_ORDER TO order_manager;
GRANT EXECUTE ON ck_order TO order_manager;
GRANT EXECUTE ON count_customer TO order_manager;
GRANT EXECUTE ON pk_order TO order_manager;
GRANT EXECUTE ON count_product TO order_manager;
GRANT EXECUTE ON check_price TO order_manager;
GRANT EXECUTE ON order_count TO order_manager;
GRANT EXECUTE ON check_stock TO order_manager;
GRANT EXECUTE ON order_stock TO order_manager;
GRANT EXECUTE ON input_notice TO chat_manager;
```

프로시저, 함수 접근 권한 부여

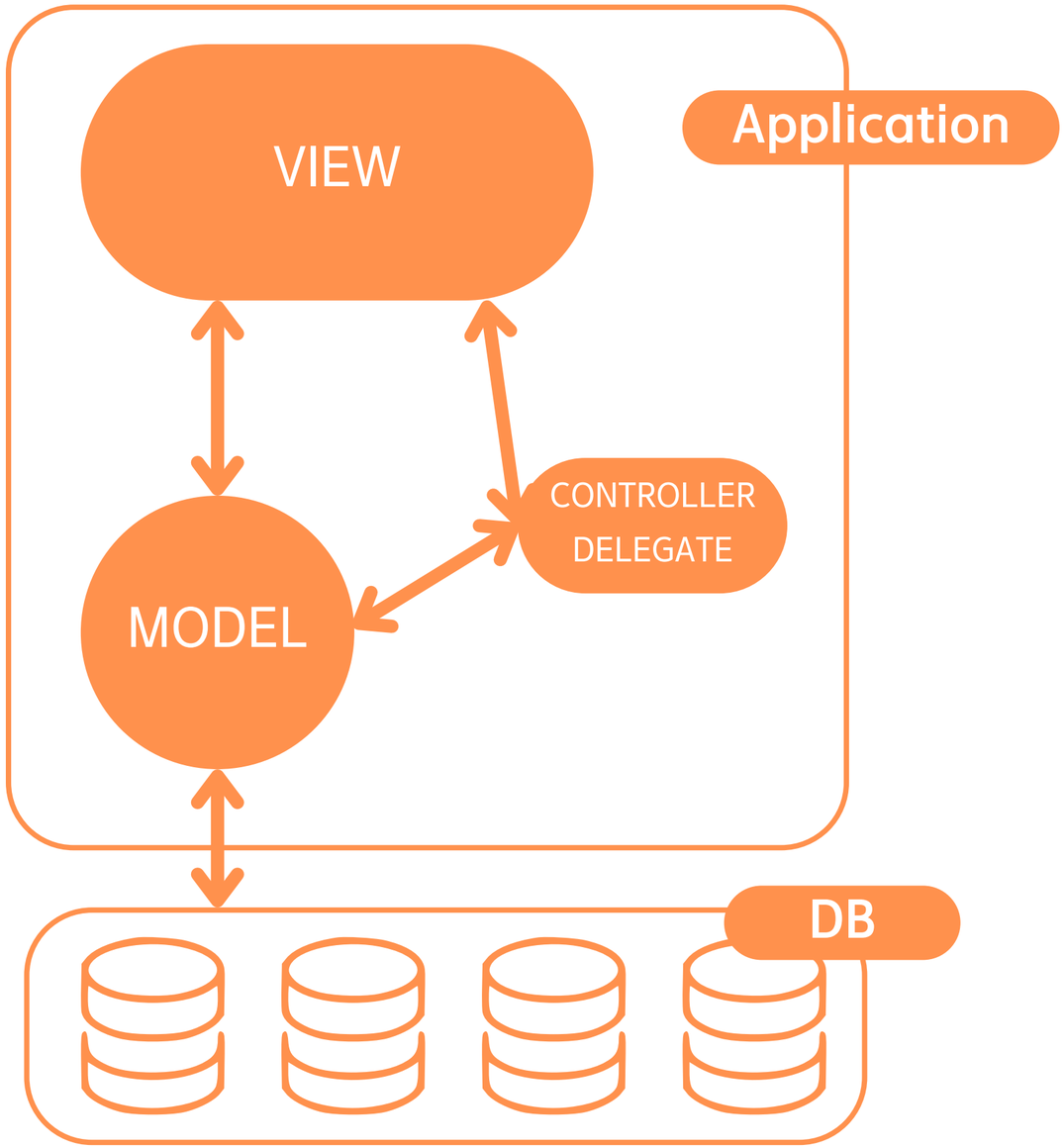
# 2-3. 3차 프로젝트
## - DB 설계_권한 설정

```
create or replace PROCEDURE INPUT_CUSTOMER
(
    ck          IN CUSTOMER_TABLE.CUSTOMER_KEY%TYPE,
    clinic      IN CUSTOMER_TABLE.CLINIC_NAME%TYPE,
    license     IN CUSTOMER_TABLE.LICENSE_NUMBER%TYPE,
    dentist     IN CUSTOMER_TABLE.DENTIST_NAME%TYPE,
    number_c IN CUSTOMER_TABLE.PHONE_NUMBER%TYPE,
    order_count    IN CUSTOMER_TABLE.ORDER_COUNT%TYPE
)
IS
BEGIN
    INSERT INTO CUSTOMER_TABLE(CUSTOMER_KEY, CLINIC_NA
    VALUES  (ck, clinic, license, dentist, number_c, o
END INPUT_CUSTOMER;
```

프로시저 예시

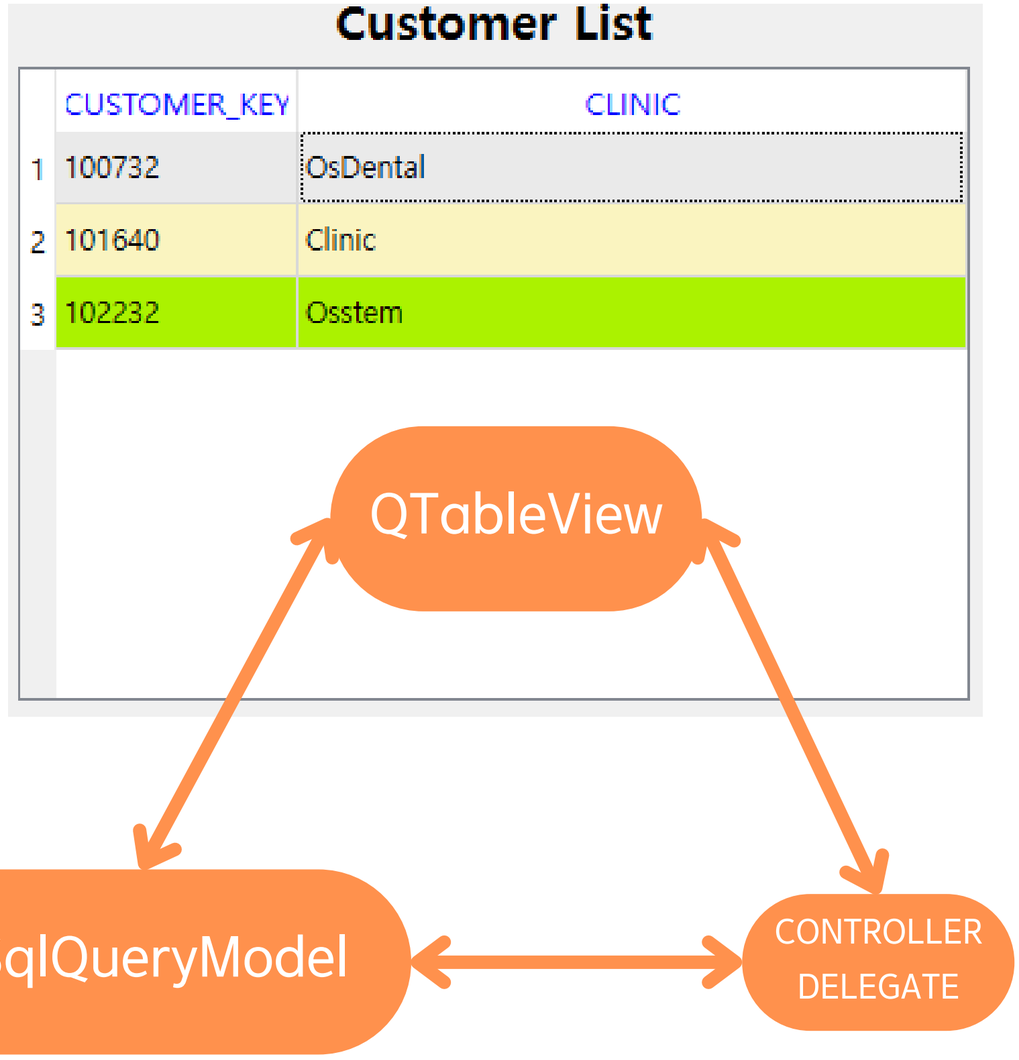
- 특정 계정에 데이터 추가를 위하여 직접적인 INSERT 문의 권한까지 부여하는 것을 최소화
- 필요할 경우 DBA에서 특정 행위에 대한 프로시저를 정의하고 해당 프로시저에 대한 권한을 부여하여, DBA가 지정한 방식으로만 데이터를 추가할 수 있도록 제한
- 데이터 보안과 직접적인 시스템 DB 접근에 대한 안전 확보

# 2-3. 3차 프로젝트
## - DB에 UI 적용_MVC패턴




- DB에 있는 데이터를 UI로 표현하기 위해, MVC 패턴을 사용
- 실제 물리적인 데이터는 DB에 저장
- Application에 있는 Model이 데이터를 가지고 있는 형태
- View는 모델에 있는 데이터를 사용자가 눈으로 볼 수 있게 화면에 표시하는 역할
- Qt에서 Controller는 Delegate를 사용할 수 있음
- Delegate를 활용하여 Model과 View사이에서 특정데이터에 대한 동작을 수행할 수 있음

# 2-3. 3차 프로젝트
## - DB에 UI 적용_MVC패턴

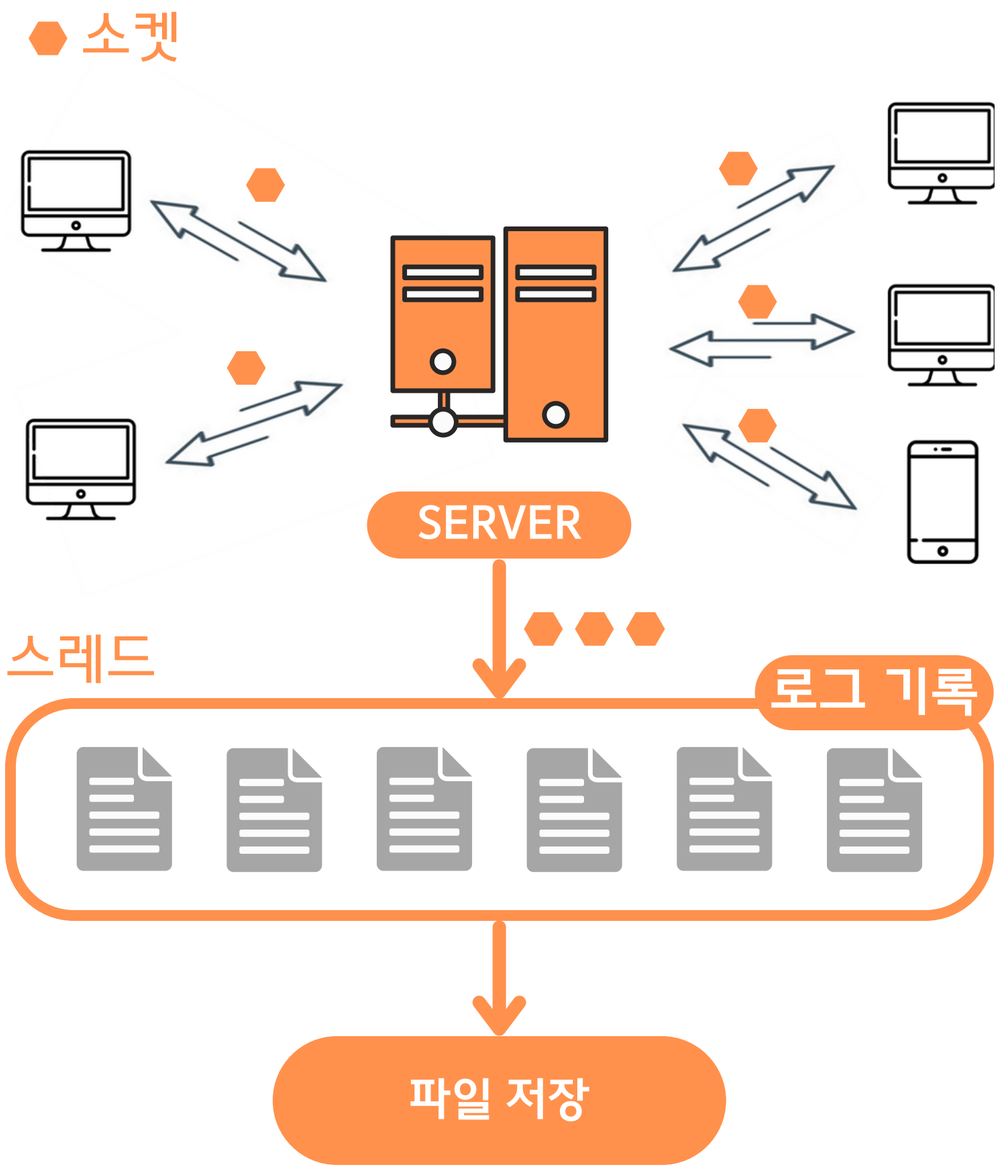



- Model은 QSqlQueryModel을 기본으로 사용
- View는 QTableView를 사용

- Model의 데이터에 따라 동적으로 View의 색상을 바꾸고자 DELEGATE를 사용

# 2-3. 3차 프로젝트

## - 로그 저장_생산자-소비자 패턴




```
connect(ui->logSaveButton, SIGNAL(clicked()),
        logSaveThread, SLOT(saveData()), Qt::QueuedConnection);
```

- 로그를 기록함에 있어서, QueuedConnection을 적용함으로써 Queue형태로 시그널을 처리
- 서버에서는 계속해서 소켓을 읽으면서, 로그 기록을 만들어가고, 사용자가 ui->logSaveButton을 누르면, 로그를 저장하는 시그널이 쌓이게 됨 (생산)
- 스레드가 돌아가면서 차례로 시그널을 처리(소비)

# 3

# 결론

## 3-1. 보완 사항 및 발전 방향

# 3-1. 보완 사항 및 발전 방향

## 보완 사항

- 책임 중심의 설계를 정확하게 구현하기 위해, 인터페이스 함수 등을 적절하게 활용하여 패턴을 적용할 필요가 있음
- Qt에서 ORACLE DB에 접속하기 위해 사용한 ODBC 드라이버는 배포 시 실행될 PC에도 같은 드라이버 환경이 있어야만 실행 가능한 문제 발생
- DB에서 한글 입력 시, 깨지는 문제 해결 필요

## 발전 방향

- OCI 등의 방법을 통해, 배포 시에도 추가적인 오라클 환경설정이 필요 없도록 해결 가능
- 고객 계정에 대한 DB권한을 적절하게 적용하여, 고객의 회원가입, 주문 기능을 추가할 수 있음