Student Management Program with Oracle & C

[프로젝트 계획]

1. 직원 or 학생 or 고객 or 주문 내역 등 정보 관리 시스템

   목표 : 학생 or 직원 or 고객 or 주문 내역 등 정보를 관리할 수 있는 시스템으로, 추가, 조회, 수정, 삭제 기능을 제공

   기술 스택 :

   - 데이터베이스 엔진 : Oracle
   - 프로그래밍 언어 : C
   - 클라이언트 라이브러리 : Oracle Call Interface (OCI)

     OCI는 Oracle 데이터베이스와 상호 작용하기 위한 고성능 C API이다. OCI를 사용하면 C 및 C++ 애플리케이션에서 Oracle 데이터베이스의 강력한 기능을 직접 활용할 수 있다. OCI는 SQL믄 실행, PL/SQL 호출, 데이터베이스 연결 관리 등 다양한 데이터베이스 작업을 수행할 수 있도록 한다.

2. 직원 or 학생 or 고객 or 주문 내역 등 정보 관리 시스템

   목표 : 학생 or 직원 or 고객 or 주문 내역 등 정보를 관리할 수 있는 시스템으로, 추가, 조회, 수정, 삭제 기능을 제공

   기술 스택 :

   - 데이터베이스 엔진 : Oracle
   - 프로그래밍 언어 : Python
   - 클라이언트 라이브러리 : cx_ORACLE

     cx_ORACLE은 Python에서 Oracle 데이터베이스에 접속하고 상호작용 할 수 있도록 지원하는 라이브러리이다. 이 라이브러리는 SQL문 실행, PL/SQL 호출, 데이터베이스 연결 관리 등 다양한 데이터베이스 작업을 수행할 수 있는 기능을 제공한다.

3. 채팅 프로그램

   목표 : 사용자 간의 실시간 메시지 전송 및 수신 기능을 제공하는 채팅 프로그램

   기술 스택 :

   - 데이터베이스 엔진 : Oracle
   - 프로그래밍 언어 : C / Python
   - 클라이언트 라이브러리 : Oracle Call Interface (OCI) / cx_ORACLE
   - 네트워킹 라이브러리 : POSIX sockets (리눅스/유닉스) / socket

개발 방법 :

1. 데이터베이스 설정 : 메시지를 저장할 테이블을 생성한다.
2. 서버 코드 작성
    - 서버 소켓을 생성하고 클라이언트 연결을 수락한다.
    - 클라이언트로부터 메시지를 받아 데이터베이스에 저장하고, 모든 클라이언트에게 메시지를 브로드캐스트한다.
    1. OCI 라이브러리 / cx_ORACLE 라이브러리 설정
    2. 서버 소켓 생성 및 설정
    3. 클라이언트 관리
3. 클라이언트 코드 작성
    - 서버에 연결하고 메시지를 주고받는다.
    - 메시지를 입력받아 서버로 전송하고, 서버로부터 메시지를 수신하여 출력한다.
    1. 클라이언트 소켓 생성 및 연결
    2. 메시지 송수신

4. 유기견 관리 시스템 1

목표 : 유기견 정보를 관리하고, 유기견의 입양 및 보호 활동을 지원하는 시스템을 구현한다.

기술 스택 :

- 데이터베이스 엔진 : Oracle
- 프로그래밍 언어 : Python
- 클라이언트 라이브러리 : cx_Oracle

개발 방법 :

1. 데이터베이스 설정
    - Oracle 데이터베이스에 접속하여 유기견 정보를 저장할 테이블을 생성한다.
2. 환경 설정
    a. Python 설치
    b. 가상 환경 생성 및 활성화
        - 가상 환경 : Python 프로젝트를 독립적으로 실행할 수 있는 격리된 환경이다. 이를 통해 프로젝트마다 서로 다른 패키지 버전을 사용할 수 있으며, 시스템 전체에 영향을 미치지 않고 패키지를 설치하고 관리할 수 있다.
    c. cx_Oracle 라이브러리 설치
    d. Oracle Instant Client 설치

- Oracle Instant Client는 Oracle 데이터베이스에 연결하여 SQL 명령을 실행할 수 있는 경량 클라이언트 소프트웨어이다. 이를 통해 Oracle 데이터베이스에 쉽게 연결하고 상호 작용할 수 있다.

3. 프로그램 코드 작성
   - Python과 cx_Oracle을 사용하여 Oracle 데이터베이스에서 CRUD 작업을 수행

4. 유기견 관리 시스템 2

   목표 : 유기견 정보를 관리하고, 웹 인터페이스를 통해 CRUD 작업을 수행할 수 있는 시스템을 구현한다.

   기술 스택 :

   - 데이터베이스 엔진 : Oracle
   - 프로그래밍 언어 : Python
   - 웹 프레임워크 : Flask
   - 클라이언트 라이브러리 : cx_Oracle

   개발 방법 :

   1. 데이터베이스 설정 : Oracle 데이터베이스에 유기견 정보를 저장할 테이블을 생성한다.
   2. 환경 설정 : Python 가상 환경을 생성하고 Flask와 cx_Oracle 라이브러리를 설치한다.
   3. Flask 애플리케이션 작성 : Flask 애플리케이션을 작성하여 유기견 정보를 추가, 조회, 수정, 삭제할 수 있는 엔드포인트를 구현한다.
   4. HTML 템플릿 작성 : 유기견 정보를 입력하고, 수정 및 삭제할 수 있는 웹 페이지를 작성한다.

[프로젝트 진행]

학생 정보 관리 시스템

목표 : 학생 정보를 관리할 수 있는 시스템으로, 추가, 조회, 수정, 삭제 기능을 제공

기술 스택 :

- 데이터베이스 엔진 : Oracle
- 프로그래밍 언어 : C
- 클라이언트 라이브러리 : Oracle Call Interface (OCI)

  OCI는 Oracle 데이터베이스와 상호 작용하기 위한 고성능 C API이다. OCI를 사용하면 C 및 C++ 애플리케이션에서 Oracle 데이터베이스의 강력한 기능을 직접 활용할 수 있다. OCI는 SQL문 실행, PL/SQL 호출, 데이터베이스 연결 관리 등 다양한 데이터베이스 작업을 수행할 수 있도록 한다.

1. 환경 설정

   데이터베이스 설정 : Oracle 데이터베이스 설치 및 설정, 데이터베이스 스키마 및 테이블 생성

개발 환경 설정 : C언어 개발 환경 설치(GCC), OCI 설치 및 구성

2. 데이터베이스 설계

   학생 정보(학번, 이름, 나이, 전공)를 저장하기 위한 테이블을 설계한다.

```
CREATE TABLE Students (
    student_id NUMBER PRIMARY KEY,
    name VARCHAR2(100),
    age NUMBER,
    major VARCHAR2(100)
);
```

3. OCI 설정 및 기본 코드 작성

   OCI를 사용하여 Oracle 데이터베이스와 연결하고 기본 CRUD 작업을 수행하는 C 프로그램을 작성한다.

   OCI 초기화 및 연결 코드 :

```
#include <stdio.h>
#include <oci.h>
```

- oci.h : Oracle Call Interface (OCI) 라이브러리 헤더 파일을 포함한다.

오류 처리 함수 :

```
void checkerr(OCIError *errhp, sword status) {
    text errbuf[512];
    sb4 errcode = 0;

    switch (status) {
        case OCI_SUCCESS:
            break;
        case OCI_SUCCESS_WITH_INFO:
            printf("OCI_SUCCESS_WITH_INFO\n");
            break;
        case OCI_NEED_DATA:
            printf("OCI_NEED_DATA\n");
            break;
        case OCI_NO_DATA:
            printf("OCI_NO_DATA\n");
            break;
        case OCI_ERROR:
            OCIErrorGet((void *)errhp, (ub4)1, (text *)NULL, &errcode, e
rrbuf, (ub4)sizeof(errbuf), OCI_HTYPE_ERROR);
            printf("Error - %.*s\n", 512, errbuf);
            break;
```

```
        case OCI_INVALID_HANDLE:
            printf("OCI_INVALID_HANDLE\n");
            break;
        case OCI_STILL_EXECUTING:
            printf("OCI_STILL_EXECUTING\n");
            break;
        case OCI_CONTINUE:
            printf("OCI_CONTINUE\n");
            break;
        default:
            break;
    }
}
```

- checkerr 함수는 OCI 함수 호출 시 발생하는 오류를 처리하기 위한 함수이다. OCI 함수가 성공했는지 여부를 확인하고, 오류가 발생한 경우 오류 메시지를 출력한다.
- OCIErrorGet 함수는 OCI 오류 핸들을 사용하여 오류 메시지를 가져온다.

메인 함수 : OCI를 사용하여 Oracle 데이터베이스에 연결하고, 연결이 성공했는지 확인한 후, 연결을 해제한다.

- 변수 선언

```
int main() {
    OCIEnv *envhp;
    OCIError *errhp;
    OCISvcCtx *svchp;
    OCIStmt *stmthp;
    OCISession *usrhp;
    OCIAuthInfo *authp;
```

  - OCIEnv *envhp: OCI 환경 핸들이다. OCI 환경을 관리하는 데 사용된다.
  - OCIError *errhp: OCI 오류 핸들이다. 오류 처리를 위한 핸들이다.
  - OCISvcCtx *svchp: OCI 서비스 컨텍스트 핸들이다. 데이터베이스 연결을 관리하는 데 사용된다.
  - OCIStmt *stmthp: OCI 문장 핸들이다. SQL 문장을 실행하는 데 사용된다.
  - OCISession *usrhp: OCI 세션 핸들이다. 사용자 세션을 관리하는 데 사용된다.
  - OCIAuthInfo *authp: OCI 인증 정보 핸들이다. 사용자 인증 정보를 관리하는 데 사용된다.
- 데이터베이스 연결 정보 설정

```
    char *username = "minsung";
    char *password = "silcroad";
    char *dbname = "STUDENTS";
```

  - char *username: 데이터베이스 사용자 이름이다.
  - char *password: 데이터베이스 사용자 비밀번호이다.

- char *dbname: 데이터베이스 이름이다.

- OCI 환경 초기화

```
    if (OCIEnvCreate((OCIEnv **)&envhp, (ub4)OCI_THREADED | OCI_OBJEC
T, (dvoid *)0,
                      0, 0, 0, (size_t)0, (dvoid **)0)) {
        printf("OCIEnvCreate failed\n");
        return -1;
    }
```

  - OCIEnvCreate: OCI 환경을 생성한다. 성공하면 OCI_SUCCESS를 반환한다. 실패하면 오류 메시지를 출력하고 프로그램을 종료한다.

- OCI 핸들 할당

```
    OCIHandleAlloc((dvoid *)envhp, (dvoid **)&errhp, OCI_HTYPE_ERROR,
(size_t)0, (dvoid **)0);
    OCIHandleAlloc((dvoid *)envhp, (dvoid **)&svchp, OCI_HTYPE_SVCCT
X, (size_t)0, (dvoid **)0);
    OCIHandleAlloc((dvoid *)envhp, (dvoid **)&authp, OCI_HTYPE_AUTHIN
FO, (size_t)0, (dvoid **)0);
```

  - OCIHandleAlloc : 필요한 OCI 핸들을 할당한다. 각 핸들은 환경 핸들(envhp)에 연결된다.
    - errhp : 오류 처리를 위한 핸들이다.
    - svchp : 서비스 컨텍스트 핸들이다.
    - authp : 인증 정보 핸들이다.

- 인증 정보 설정

```
    OCIAttrSet((dvoid *)authp, OCI_HTYPE_AUTHINFO, (dvoid *)username,
(ub4)strlen(username), OCI_ATTR_USERNAME, errhp);
    OCIAttrSet((dvoid *)authp, OCI_HTYPE_AUTHINFO, (dvoid *)password,
(ub4)strlen(password), OCI_ATTR_PASSWORD, errhp);
```

  - OCIAttrSet : 인증 정보를 설정한다. 사용자 이름과 비밀번호를 설정한다.
    - 첫 번째 OCIAttrSet : 사용자 이름을 설정한다.
    - 두 번째 OCIAttrSet : 사용자 비밀번호를 설정한다.

- 데이터베이스 연결

```
    if (OCILogon2(envhp, errhp, &svchp, authp, OCI_LOGON2_STMTCACHE |
OCI_LOGON2_CPOOL)) {
        checkerr(errhp, OCI_ERROR);
        return -1;
    }
```

- OCILogon2 : 데이터베이스에 연결한다. 성공하면 OCI_SUCCESS를 반환한다. 실패하면 오류 메시지를 출력하고 프로그램을 종료한다.

- 연결 성공 메시지 출력

```
    printf("Connected to the database\n");
```

  - 데이터베이스에 성공적으로 연결되면 메시지를 출력한다.

- 자원 해제 및 로그아웃

```
    // Release handles and logout
    OCILogoff(svchp, errhp);
    OCIHandleFree((dvoid *)authp, OCI_HTYPE_AUTHINFO);
    OCIHandleFree((dvoid *)errhp, OCI_HTYPE_ERROR);
    OCIHandleFree((dvoid *)svchp, OCI_HTYPE_SVCCTX);
    OCIEnvDestroy(envhp);

    return 0;
}
```

  - `OCILogoff` : 데이터베이스에서 로그아웃한다.
  - `OCIHandleFree` : 사용한 OCI 핸들을 해제한다.
  - `OCIEnvDestroy` : OCI 환경을 제거한다.

4. CRUD 기능 구현

   Create : OCI를 사용하여 Students 테이블에 새로운 레코드를 삽입한다.

   - 함수 시그니처

```
void addStudent(OCISvcCtx *svchp, OCIError *errhp, int student_id, ch
ar *name, int age, char *major)
```

     - OCISvcCtx *svchp : 데이터베이스 서비스 컨텍스트 핸들이다. 이 핸들은 데이터베이스 연결 정보를 포함한다.
     - OCIError *errhp : 오류 핸들이다. 오류 정보를 저장하는 데 사용된다.
     - int student_id : 삽입할 학생의 학번
     - char *name : 삽입할 학생의 이름
     - int age : 삽입할 학생의 나이
     - char *major : 삽입할 학생의 전공

   - SQL 문장 준비

```
    OCIStmt *stmthp;
    const char *sql = "INSERT INTO Students (student_id, name, age, m
ajor) VALUES (:1, :2, :3, :4)";
```

- OCIStmt *stmthp : SQL 문장을 실행하는 데 사용되는 OCI 문장 핸들이다.
- const char *sql : 삽입할 SQL문이다. `:1, :2, :3, :4`는 바인드 변수이다.

- OCI 문장 핸들 할당

```
    OCIHandleAlloc((dvoid *)envhp, (dvoid **)&stmthp, OCI_HTYPE_STMT,
(size_t)0, (dvoid **)0);
```

  - OCIHandleAlloc : OCI 문장 핸들을 할당한다. envhp 환경 핸들을 사용하여 문장 핸들을 할당한다.
  - stmthp : SQL 문장을 실행하는 데 사용된다.

- SQL 문장 준비

```
    OCIStmtPrepare(stmthp, errhp, (text *)sql, (ub4)strlen(sql), (ub
4)OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT);
```

  - OCIStmtPrepare : SQL 문을 준비한다. semthp 문장 핸들을 사용하여 sql 문자열을 준비한다.

- 바인드 변수 설정

```
    OCIBindByPos(stmthp, &bindhp[0], errhp, 1, (dvoid *)&student_id,
(sb4)sizeof(student_id), SQLT_INT, (dvoid *)0, (ub2 *)0, (ub2 *)0, 0,
(ub4 *)0, OCI_DEFAULT);
    OCIBindByPos(stmthp, &bindhp[1], errhp, 2, (dvoid *)name, (sb4)st
rlen(name) + 1, SQLT_STR, (dvoid *)0, (ub2 *)0, (ub2 *)0, 0, (ub4 *)
0, OCI_DEFAULT);
    OCIBindByPos(stmthp, &bindhp[2], errhp, 3, (dvoid *)&age, (sb4)si
zeof(age), SQLT_INT, (dvoid *)0, (ub2 *)0, (ub2 *)0, 0, (ub4 *)0, OCI
_DEFAULT);
    OCIBindByPos(stmthp, &bindhp[3], errhp, 4, (dvoid *)major, (sb4)s
trlen(major) + 1, SQLT_STR, (dvoid *)0, (ub2 *)0, (ub2 *)0, 0, (ub4
*)0, OCI_DEFAULT);
```

  - OCIBindByPos: SQL 문장에서 바인드 변수(:1, :2, :3, :4)를 실제 데이터 변수(student_id, name, age, major)에 매핑한다.
    1. :1을 student_id에 매핑합니다.
    2. :2을 name에 매핑합니다.
    3. :3을 age에 매핑합니다.
    4. :4을 major에 매핑합니다.

- SQL 문장 실행

```
    if (OCIStmtExecute(svchp, stmthp, errhp, 1, 0, (OCISnapshot *)NUL
L, (OCISnapshot *)NULL, OCI_DEFAULT) != OCI_SUCCESS) {
        checkerr(errhp, OCI_ERROR);
    } else {
```

```
        printf("Student added successfully\n");
    }
```

- OCIStmtExecute : SQL 문장을 실행한다. 성공하면 OCI_SUCCESS를 반환하고, 실패하면 오류 메시지를 출력한다.
- checkerr : 오류 발생시 호출되는 함수로, 오류 메시지를 출력한다.
- 성공시 Student added successfully를 출력한다.

- 문장 핸들 해제

```
OCIStmtRelease(stmthp, errhp, (text *)NULL, 0, OCI_DEFAULT);
```

- OCIStmtRelease : 사용한 문장 핸들을 해제하여 자원을 반환한다.

Read : OCI를 사용하여 Students 테이블에서 특정 student_id에 해당하는 학생 정보를 가져와 출력한다.

- 함수 시그니처

```
void getStudent(OCISvcCtx *svchp, OCIError *errhp, int student_id)
```

- OCISvcCtx *svchp : 데이터베이스 서비스 컨텍스트 핸들이다. 이 핸들은 데이터베이스 연결 정보를 포함한다.
- OCIError *errhp : 오류 핸들이다. 오류 정보를 저장하는 데 사용된다.
- int student_id : 조회할 학생의 학번이다.

- 변수 선언

```
    OCIStmt *stmthp;
    OCIDefine *defnp[4];
    int id;
    char name[100];
    int age;
    char major[100];
```

- OCIStmt *stmthp: SQL 문장을 실행하는 데 사용되는 OCI 문장 핸들이다.
- OCIDefine *defnp[4]: 컬럼 값을 정의하는 데 사용되는 정의 핸들 배열이다.
- int id: 조회된 학생의 학번을 저장할 변수이다.
- char name[100]: 조회된 학생의 이름을 저장할 배열이다.
- int age: 조회된 학생의 나이를 저장할 변수이다.
- char major[100]: 조회된 학생의 전공을 저장할 배열이다.

- SQL 문장 준비

```
    const char *sql = "SELECT student_id, name, age, major FROM Stude
nts WHERE student_id = :1";
```

- const char *sql : 조회할 SQL 문이다. :1는 바인드 변수이다.

- OCI 문장 핸들 할당

```
    OCIHandleAlloc((dvoid *)envhp, (dvoid **)&stmthp, OCI_HTYPE_STMT,
(size_t)0, (dvoid **)0);
```

  - OCIHandleAlloc: OCI 문장 핸들을 할당한다. envhp 환경 핸들을 사용하여 문장 핸들을 할당한다.
  - stmthp는 SQL 문장을 실행하는 데 사용된다.

- SQL 문장 준비

```
    OCIStmtPrepare(stmthp, errhp, (text *)sql, (ub4)strlen(sql), (ub
4)OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT);
```

  - OCIStmtPrepare : SQL 문을 준비한다. stmthp 문장 핸들을 사용하여 sql 문자열을 준비한다.

- 바인드 변수 설정

```
    OCIBindByPos(stmthp, &bindhp[0], errhp, 1, (dvoid *)&student_id,
(sb4)sizeof(student_id), SQLT_INT, (dvoid *)0, (ub2 *)0, (ub2 *)0, 0,
(ub4 *)0, OCI_DEFAULT);
```

  - OCIBindByPos : SQL 문장에서 바인드 변수 :1을 실제 데이터 변수 student_id에 매핑한다.

- 정의 핸들 설정

```
    OCIDefineByPos(stmthp, &defnp[0], errhp, 1, (dvoid *)&id, (sb4)si
zeof(id), SQLT_INT, (dvoid *)0, (ub2 *)0, (ub2 *)0, OCI_DEFAULT);
    OCIDefineByPos(stmthp, &defnp[1], errhp, 2, (dvoid *)name, (sb4)s
izeof(name), SQLT_STR, (dvoid *)0, (ub2 *)0, (ub2 *)0, OCI_DEFAULT);
    OCIDefineByPos(stmthp, &defnp[2], errhp, 3, (dvoid *)&age, (sb4)s
izeof(age), SQLT_INT, (dvoid *)0, (ub2 *)0, (ub2 *)0, OCI_DEFAULT);
    OCIDefineByPos(stmthp, &defnp[3], errhp, 4, (dvoid *)major, (sb4)
sizeof(major), SQLT_STR, (dvoid *)0, (ub2 *)0, (ub2 *)0, OCI_DEFAUL
T);
```

  - OCIDefineByPos : SQL 문장에서 컬럼 값을 저장할 변수를 정의한다.
    1. 첫 번째 컬럼(student_id)을 id에 매핑한다.
    2. 두 번째 컬럼(name)을 name에 매핑한다.
    3. 세 번째 컬럼(age)을 age에 매핑한다.
    4. 네 번째 컬럼(major)을 major에 매핑한다.

- SQL 문장 실행 및 데이터 가져오기

```
if (OCIStmtExecute(svchp, stmthp, errhp, 1, 0, (OCISnapshot *)NULL,
(OCISnapshot *)NULL, OCI_DEFAULT) == OCI_SUCCESS) {
    while (OCIStmtFetch2(stmthp, errhp, 1, OCI_DEFAULT, 1, OCI_DEFAUL
T) == OCI_SUCCESS) {
```

```
        printf("ID: %d, Name: %s, Age: %d, Major: %s\n", id, name, ag
e, major);
    }
} else {
    checkerr(errhp, OCI_ERROR);
}
```

- OCIStmtExecute: SQL 문장을 실행한다. 성공하면 OCI_SUCCESS를 반환하고, 실패하면 오류 메시지를 출력한다.
- OCIStmtFetch2: 실행된 SQL 문장에서 결과를 한 행씩 가져온다. 성공하면 OCI_SUCCESS를 반환하고, 가져온 데이터를 출력한다.
- printf: 가져온 데이터를 출력한다.
- checkerr: 오류 발생 시 호출되는 함수로, 오류 메시지를 출력한다.

- 문장 핸들 해제

```
OCIStmtRelease(stmthp, errhp, (text *)NULL, 0, OCI_DEFAULT);
```

- OCIStmtRelease: 사용한 문장 핸들을 해제하여 자원을 반환한다.

Update : OCI를 사용하여 Students 테이블에서 특정 student_id에 해당하는 학생의 이름, 나이, 전공 정보를 업데이트한다.

- 함수 시그니처

```
void updateStudent(OCISvcCtx *svchp, OCIError *errhp, int student_id,
char *name, int age, char *major)
```

- OCISvcCtx *svchp: 데이터베이스 서비스 컨텍스트 핸들이다. 이 핸들은 데이터베이스 연결 정보를 포함한다.
- OCIError *errhp: 오류 핸들이다. 오류 정보를 저장하는 데 사용된다.
- int student_id: 업데이트할 학생의 학번이다.
- char *name: 업데이트할 학생의 이름이다.
- int age: 업데이트할 학생의 나이이다.
- char *major: 업데이트할 학생의 전공이다.

- SQL 문장 준비

```
    OCIStmt *stmthp;
    const char *sql = "UPDATE Students SET name = :1, age = :2, major
= :3 WHERE student_id = :4";
```

- `OCIStmt *stmthp`: SQL 문장을 실행하는 데 사용되는 OCI 문장 핸들이다.
- `const char *sql`: 업데이트할 SQL 문이다. `:1`, `:2`, `:3`, `:4`는 바인드 변수이다.

- OCI 문장 핸들 할당

```
    OCIHandleAlloc((dvoid *)envhp, (dvoid **)&stmthp, OCI_HTYPE_STMT,
(size_t)0, (dvoid **)0);
```

- OCIHandleAlloc: OCI 문장 핸들을 할당한다. envhp 환경 핸들을 사용하여 문장 핸들을 할당한다.
- stmthp는 SQL 문장을 실행하는 데 사용된다.

- SQL 문장 준비

```
    OCIStmtPrepare(stmthp, errhp, (text *)sql, (ub4)strlen(sql), (ub
4)OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT);
```

- OCIStmtPrepare : SQL 문을 준비한다. stmthp 문장 핸들을 사용하여 sql 문자열을 준비한다.

- 바인드 변수 설정

```
    OCIBindByPos(stmthp, &bindhp[0], errhp, 1, (dvoid *)name, (sb4)st
rlen(name) + 1, SQLT_STR, (dvoid *)0, (ub2 *)0, (ub2 *)0, 0, (ub4 *)
0, OCI_DEFAULT);
    OCIBindByPos(stmthp, &bindhp[1], errhp, 2, (dvoid *)&age, (sb4)si
zeof(age), SQLT_INT, (dvoid *)0, (ub2 *)0, (ub2 *)0, 0, (ub4 *)0, OCI
_DEFAULT);
    OCIBindByPos(stmthp, &bindhp[2], errhp, 3, (dvoid *)major, (sb4)s
trlen(major) + 1, SQLT_STR, (dvoid *)0, (ub2 *)0, (ub2 *)0, 0, (ub4
*)0, OCI_DEFAULT);
    OCIBindByPos(stmthp, &bindhp[3], errhp, 4, (dvoid *)&student_id,
(sb4)sizeof(student_id), SQLT_INT, (dvoid *)0, (ub2 *)0, (ub2 *)0, 0,
(ub4 *)0, OCI_DEFAULT);
```

- OCIBindByPos : SQL 문장에서 바인드 변수 (:1, :2, :3, :4)를 실제 데이터 변수 (name. age, major, student_id)에 매핑한다.

- SQL 문장 실행

```
    if (OCIStmtExecute(svchp, stmthp, errhp, 1, 0, (OCISnapshot *)NUL
L, (OCISnapshot *)NULL, OCI_DEFAULT) != OCI_SUCCESS) {
        checkerr(errhp, OCI_ERROR);
    } else {
        printf("Student updated successfully\n");
    }
```

- OCIStmtExecute: SQL 문장을 실행한다. 성공하면 OCI_SUCCESS를 반환하고, 실패하면 오류 메시지를 출력한다.
- checkerr: 오류 발생 시 호출되는 함수로, 오류 메시지를 출력한다.
- 성공 시 "Student updated successfully" 메시지를 출력한다.

- 문장 핸들 해제

```
    OCIStmtRelease(stmthp, errhp, (text *)NULL, 0, OCI_DEFAULT);
```

- OCIStmtRelease : 사용한 문장 핸들을 해제하여 자원을 반환한다.

Delete : OCI를 사용하여 Students 테이블에서 특정 student_id에 해당하는 학생 정보를 삭제한다.

- 함수 시그니처

```
void deleteStudent(OCISvcCtx *svchp, OCIError *errhp, int student_id)
```

  - OCISvcCtx *svchp: 데이터베이스 서비스 컨텍스트 핸들이다. 이 핸들은 데이터베이스 연결 정보를 포함한다.
  - OCIError *errhp: 오류 핸들이다. 오류 정보를 저장하는 데 사용된다.
  - int student_id: 삭제할 학생의 학번이다.

- SQL 문장 준비

```
    OCIStmt *stmthp;
    const char *sql = "DELETE FROM Students WHERE student_id = :1";
```

  - OCIStmt *stmthp : SQL 문장을 실행하는 데 사용되는 OCI 문장 핸들이다.
  - const char *sql : 삭제할 SQL 문이다. :1는 바인드 변수이다.

- OCI 문장 핸들 할당

```
    OCIHandleAlloc((dvoid *)envhp, (dvoid **)&stmthp, OCI_HTYPE_STMT,
(size_t)0, (dvoid **)0);
```

  - OCIHandleAlloc: OCI 문장 핸들을 할당한다. envhp 환경 핸들을 사용하여 문장 핸들을 할당한다.
  - stmthp는 SQL 문장을 실행하는 데 사용된다.

- SQL 문장 준비

```
    OCIStmtPrepare(stmthp, errhp, (text *)sql, (ub4)strlen(sql), (ub
4)OCI_NTV_SYNTAX, (ub4)OCI_DEFAULT);
```

  - OCIStmtPrepare : SQL 문을 준비한다. stmthp 문장 핸들을 사용하여 sql 문자열을 준비한다.

- 바인드 변수 설정

```
    OCIBindByPos(stmthp, &bindhp[0], errhp, 1, (dvoid *)&student_id,
(sb4)sizeof(student_id), SQLT_INT, (dvoid *)0, (ub2 *)0, (ub2 *)0, 0,
(ub4 *)0, OCI_DEFAULT);
```

  - OCIBindByPos: SQL 문장에서 바인드 변수 :1을 실제 데이터 변수 student_id에 매핑한다.

- SQL 문장 실행

```
    if (OCIStmtExecute(svchp, stmthp, errhp, 1, 0, (OCISnapshot *)NUL
L, (OCISnapshot *)NULL, OCI_DEFAULT) != OCI_SUCCESS) {
        checkerr(errhp, OCI_ERROR);
```

```
    } else {
        printf("Student deleted successfully\n");
    }
```

- OCIStmtExecute: SQL 문장을 실행한다. 성공하면 OCI_SUCCESS를 반환하고, 실패하면 오류 메시지를 출력한다.
- checkerr: 오류 발생 시 호출되는 함수로, 오류 메시지를 출력한다.
- 성공 시 "Student deleted successfully" 메시지를 출력한다.

- 문장 핸들 해제

```
    OCIStmtRelease(stmthp, errhp, (text *)NULL, 0, OCI_DEFAULT);
```

- OCIStmtRelease : 사용한 문장 핸들을 해제하여 자원을 반환한다.

최종 코드 :

```
#include <stdio.h>
#include <oci.h>

// 오류 처리 함수
void checkerr(OCIError *errhp, sword status) {
    text errbuf[512];
    sb4 errcode = 0;

    switch (status) {
        case OCI_SUCCESS:
            break;
        case OCI_SUCCESS_WITH_INFO:
            printf("OCI_SUCCESS_WITH_INFO\n");
            break;
        case OCI_NEED_DATA:
            printf("OCI_NEED_DATA\n");
            break;
        case OCI_NO_DATA:
            printf("OCI_NO_DATA\n");
            break;
        case OCI_ERROR:
            OCIErrorGet((void *)errhp, (ub4)1, (text *)NULL, &errcode, e
rrbuf, (ub4)sizeof(errbuf), OCI_HTYPE_ERROR);
            printf("Error - %.*s\n", 512, errbuf);
            break;
        case OCI_INVALID_HANDLE:
            printf("OCI_INVALID_HANDLE\n");
            break;
        case OCI_STILL_EXECUTING:
            printf("OCI_STILL_EXECUTING\n");
            break;
```

```
        case OCI_CONTINUE:
            printf("OCI_CONTINUE\n");
            break;
        default:
            break;
    }
}

int main() {
    OCIEnv *envhp;
    OCIError *errhp;
    OCISvcCtx *svchp;
    OCIStmt *stmthp;
    OCISession *usrhp;
    OCIAuthInfo *authp;

    char *username = "minsung";
    char *password = "silcroad";
    char *dbname = "STUDENTS";

    if (OCIEnvCreate((OCIEnv **)&envhp, (ub4)OCI_THREADED | OCI_OBJECT,
(dvoid *)0,
                     0, 0, 0, (size_t)0, (dvoid **)0)) {
        printf("OCIEnvCreate failed\n");
        return -1;
    }

    OCIHandleAlloc((dvoid *)envhp, (dvoid **)&errhp, OCI_HTYPE_ERROR, (s
ize_t)0, (dvoid **)0);
    OCIHandleAlloc((dvoid *)envhp, (dvoid **)&svchp, OCI_HTYPE_SVCCTX,
(size_t)0, (dvoid **)0);
    OCIHandleAlloc((dvoid *)envhp, (dvoid **)&authp, OCI_HTYPE_AUTHINFO,
(size_t)0, (dvoid **)0);

    OCIAttrSet((dvoid *)authp, OCI_HTYPE_AUTHINFO, (dvoid *)username, (u
b4)strlen(username), OCI_ATTR_USERNAME, errhp);
    OCIAttrSet((dvoid *)authp, OCI_HTYPE_AUTHINFO, (dvoid *)password, (u
b4)strlen(password), OCI_ATTR_PASSWORD, errhp);

    if (OCILogon2(envhp, errhp, &svchp, authp, OCI_LOGON2_STMTCACHE | OC
I_LOGON2_CPOOL)) {
        checkerr(errhp, OCI_ERROR);
        return -1;
    }

    printf("Connected to the database\n");

    // 학생 정보 추가
```

```
    addStudent(svchp, errhp, 1, "Minsung", 20, "Applied Physics");

    // 학생 정보 조회
    getStudent(svchp, errhp, 1);

    // 학생 정보 수정
    updateStudent(svchp, errhp, 1, "Minsung", 21, "Computer Science");

        // 학생 정보 조회
    getStudent(svchp, errhp, 1);

    // 학생 정보 삭제
    deleteStudent(svchp, errhp, 1);

    // 자원 해제 및 로그아웃
    OCILogoff(svchp, errhp);
    OCIHandleFree((dvoid *)authp, OCI_HTYPE_AUTHINFO);
    OCIHandleFree((dvoid *)errhp, OCI_HTYPE_ERROR);
    OCIHandleFree((dvoid *)svchp, OCI_HTYPE_SVCCTX);
    OCIEnvDestroy(envhp);

    return 0;
}

void addStudent(OCISvcCtx *svchp, OCIError *errhp, int student_id, char
*name, int age, char *major) {
    OCIStmt *stmthp;
    const char *sql = "INSERT INTO Students (student_id, name, age, majo
r) VALUES (:1, :2, :3, :4)";

    OCIHandleAlloc((dvoid *)envhp, (dvoid **)&stmthp, OCI_HTYPE_STMT, (s
ize_t)0, (dvoid **)0);
    OCIStmtPrepare(stmthp, errhp, (text *)sql, (ub4)strlen(sql), (ub4)OC
I_NTV_SYNTAX, (ub4)OCI_DEFAULT);

    OCIBindByPos(stmthp, &bindhp[0], errhp, 1, (dvoid *)&student_id, (sb
4)sizeof(student_id), SQLT_INT, (dvoid *)0, (ub2 *)0, (ub2 *)0, 0, (ub4
*)0, OCI_DEFAULT);
    OCIBindByPos(stmthp, &bindhp[1], errhp, 2, (dvoid *)name, (sb4)strle
n(name) + 1, SQLT_STR, (dvoid *)0, (ub2 *)0, (ub2 *)0, 0, (ub4 *)0, OCI_
DEFAULT);
    OCIBindByPos(stmthp, &bindhp[2], errhp, 3, (dvoid *)&age, (sb4)sizeo
f(age), SQLT_INT, (dvoid *)0, (ub2 *)0, (ub2 *)0, 0, (ub4 *)0, OCI_DEFAU
LT);
    OCIBindByPos(stmthp, &bindhp[3], errhp, 4, (dvoid *)major, (sb4)strl
en(major) + 1, SQLT_STR, (dvoid *)0, (ub2 *)0, (ub2 *)0, 0, (ub4 *)0, OC
I_DEFAULT);
```

```
    if (OCIStmtExecute(svchp, stmthp, errhp, 1, 0, (OCISnapshot *)NULL,
(OCISnapshot *)NULL, OCI_DEFAULT) != OCI_SUCCESS) {
        checkerr(errhp, OCI_ERROR);
    } else {
        printf("Student added successfully\n");
    }

    OCIStmtRelease(stmthp, errhp, (text *)NULL, 0, OCI_DEFAULT);
}

void getStudent(OCISvcCtx *svchp, OCIError *errhp, int student_id) {
    OCIStmt *stmthp;
    OCIDefine *defnp[4];
    int id;
    char name[100];
    int age;
    char major[100];

    const char *sql = "SELECT student_id, name, age, major FROM Students
WHERE student_id = :1";

    OCIHandleAlloc((dvoid *)envhp, (dvoid **)&stmthp, OCI_HTYPE_STMT, (s
ize_t)0, (dvoid **)0);
    OCIStmtPrepare(stmthp, errhp, (text *)sql, (ub4)strlen(sql), (ub4)OC
I_NTV_SYNTAX, (ub4)OCI_DEFAULT);

    OCIBindByPos(stmthp, &bindhp[0], errhp, 1, (dvoid *)&student_id, (sb
4)sizeof(student_id), SQLT_INT, (dvoid *)0, (ub2 *)0, (ub2 *)0, 0, (ub4
*)0, OCI_DEFAULT);

    OCIDefineByPos(stmthp, &defnp[0], errhp, 1, (dvoid *)&id, (sb4)sizeo
f(id), SQLT_INT, (dvoid *)0, (ub2 *)0, (ub2 *)0, OCI_DEFAULT);
    OCIDefineByPos(stmthp, &defnp[1], errhp, 2, (dvoid *)name, (sb4)size
of(name), SQLT_STR, (dvoid *)0, (ub2 *)0, (ub2 *)0, OCI_DEFAULT);
    OCIDefineByPos(stmthp, &defnp[2], errhp, 3, (dvoid *)&age, (sb4)size
of(age), SQLT_INT, (dvoid *)0, (ub2 *)0, (ub2 *)0, OCI_DEFAULT);
    OCIDefineByPos(stmthp, &defnp[3], errhp, 4, (dvoid *)major, (sb4)siz
eof(major), SQLT_STR, (dvoid *)0, (ub2 *)0, (ub2 *)0, OCI_DEFAULT);

    if (OCIStmtExecute(svchp, stmthp, errhp, 1, 0, (OCISnapshot *)NULL,
(OCISnapshot *)NULL, OCI_DEFAULT) == OCI_SUCCESS) {
        while (OCIStmtFetch2(stmthp, errhp, 1, OCI_DEFAULT, 1, OCI_DEFAU
LT) == OCI_SUCCESS) {
            printf("ID: %d, Name: %s, Age: %d, Major: %s\n", id, name, a
ge, major);
        }
    } else {
        checkerr(errhp, OCI_ERROR);
```

```
    }

    OCIStmtRelease(stmthp, errhp, (text *)NULL, 0, OCI_DEFAULT);
}

void updateStudent(OCISvcCtx *svchp, OCIError *errhp, int student_id, ch
ar *name, int age, char *major) {
    OCIStmt *stmthp;
    const char *sql = "UPDATE Students SET name = :1, age = :2, major =
:3 WHERE student_id = :4";

    OCIHandleAlloc((dvoid *)envhp, (dvoid **)&stmthp, OCI_HTYPE_STMT, (s
ize_t)0, (dvoid **)0);
    OCIStmtPrepare(stmthp, errhp, (text *)sql, (ub4)strlen(sql), (ub4)OC
I_NTV_SYNTAX, (ub4)OCI_DEFAULT);

    OCIBindByPos(stmthp, &bindhp[0], errhp, 1, (dvoid *)name, (sb4)strle
n(name) + 1, SQLT_STR, (dvoid *)0, (ub2 *)0, (ub2 *)0, 0, (ub4 *)0, OCI_
DEFAULT);
    OCIBindByPos(stmthp, &bindhp[1], errhp, 2, (dvoid *)&age, (sb4)sizeo
f(age), SQLT_INT, (dvoid *)0, (ub2 *)0, (ub2 *)0, 0, (ub4 *)0, OCI_DEFAU
LT);
    OCIBindByPos(stmthp, &bindhp[2], errhp, 3, (dvoid *)major, (sb4)strl
en(major) + 1, SQLT_STR, (dvoid *)0, (ub2 *)0, (ub2 *)0, 0, (ub4 *)0, OC
I_DEFAULT);
    OCIBindByPos(stmthp, &bindhp[3], errhp, 4, (dvoid *)&student_id, (sb
4)sizeof(student_id), SQLT_INT, (dvoid *)0, (ub2 *)0, (ub2 *)0, 0, (ub4
*)0, OCI_DEFAULT);

    if (OCIStmtExecute(svchp, stmthp, errhp, 1, 0, (OCISnapshot *)NULL,
(OCISnapshot *)NULL, OCI_DEFAULT) != OCI_SUCCESS) {
        checkerr(errhp, OCI_ERROR);
    } else {
        printf("Student updated successfully\n");
    }

    OCIStmtRelease(stmthp, errhp, (text *)NULL, 0, OCI_DEFAULT);
}

void deleteStudent(OCISvcCtx *svchp, OCIError *errhp, int student_id) {
    OCIStmt *stmthp;
    const char *sql = "DELETE FROM Students WHERE student_id = :1";

    OCIHandleAlloc((dvoid *)envhp, (dvoid **)&stmthp, OCI_HTYPE_STMT, (s
ize_t)0, (dvoid **)0);
    OCIStmtPrepare(stmthp, errhp, (text *)sql, (ub4)strlen(sql), (ub4)OC
I_NTV_SYNTAX, (ub4)OCI_DEFAULT);
```

```
    OCIBindByPos(stmthp, &bindhp[0], errhp, 1, (dvoid *)&student_id, (sb
4)sizeof(student_id), SQLT_INT, (dvoid *)0, (ub2 *)0, (ub2 *)0, 0, (ub4
*)0, OCI_DEFAULT);

    if (OCIStmtExecute(svchp, stmthp, errhp, 1, 0, (OCISnapshot *)NULL,
(OCISnapshot *)NULL, OCI_DEFAULT) != OCI_SUCCESS) {
        checkerr(errhp, OCI_ERROR);
    } else {
        printf("Student deleted successfully\n");
    }

    OCIStmtRelease(stmthp, errhp, (text *)NULL, 0, OCI_DEFAULT);
}
```