Project 1-3: Implementing DML

Due: 2025/5/20 (TUE), 11:59 P.M.

1. Project Overview

이번 프로젝트의 목표는 프로젝트 1-1 및 1-2에서 구현한 DBMS를 확장하여 다음 DML 구문들을 처리할 수 있도록 하는 것이다.

- INSERT
- DELETE
  - ✓ WHERE
- SELECT
  - ✓ WHERE
  - ✓ JOIN
  - ✓ ORDER BY
  - ✓ (Optional) GROUP BY

2. Requirements

이하의 조건들을 만족하도록 grammar.lark 파일과 run.py 파일을 작성한다.

- 2.2장에 명시된 모든 구문들을 받아 올바르게 처리할 수 있어야 한다.
- 2.3장의 구현은 **필수가 아니나**, 올바르게 구현 시 프로젝트 1-1 및 1-2에서의 감점을 만회할 수 있는 추가 점수가 **최대 20점** 부여된다.
- grammar.lark의 자세한 구현은 2.2장, 2.3장의 정의 부분을 따른다.
- 메시지 정의 문서(2025_1-3_Messages.pdf)는 DBMS가 사용할 메시지의 종류와 그 내용을 정리한 문서이다. 이를 참고하여 상황에 맞는 메시지를 프롬프트 뒤에 표시해야 한다.
  - ✓ 예시) `DB_2025-12345> SELECT has failed: No such table`
- 정의되지 않은 요소의 구현
  - ✓ 메시지 정의 문서에서 정의된 오류 유형 이외의 유형이 추가로 필요하다고 판단될 경우 해당 유형의 이름과 메시지를 직접 정의하고 보고서에 명시한다.
  - ✓ 그 외 본 문서에 정의되지 않은 요소를 구현할 경우 **MySQL**을 기준으로 구현하고 보고서에 명시한다.

- 기타 유의 사항
  - ✓ 여러 오류 유형에 포함되는 구문의 경우 해당하는 오류 메시지 중 1개를 출력한다
  - ✓ 정답 외 불필요한 출력이 추가로 나올 경우 감점된다.
  - ✓ Berkeley DB의 **SQL API 사용은 금지**된다.

2.1. Prerequisites

프로젝트 1-3에서는 이미 생성된 table 내에서 데이터를 조작하는 기능을 구현한다. 따라서 본 과제에서 사용될 DBMS는 CREATE문으로 table을 올바르게 생성할 수 있어야 한다. 아래 3개의 구문이 문제없이 실행되어 3개의 스키마가 만들어져야 한다. **아래 구문 외의 CREATE 구문은 평가하지 않는다.**

```
create table students (
    id char (10) not null,
    name char (20),
    primary key (id)
);
```

```
create table lectures (
    id int not null,
    name char (20),
    capacity int,
    primary key (id)
);
```

```
create table apply (
    s_id char (10) not null,
    l_id int not null,
    apply_date date,
    primary key (s_id, l_id),
    foreign key (s_id) references students (id),
    foreign key (l_id) references lectures (id)
);
```

DML 처리 시 primary/foreign key constraints를 따라야 하지만, 본 프로젝트에서는 그중 primary key의 not null constraint만 확인한다.

2.2. SQL Queries

본 장에서는 각 구문에 대한 정의, 처리 방법과 입출력 예제를 제공한다.

2.2.1. INSERT

- 정의

```
insert into table_name [(col_name1, col_name2, …)] values(value1, value2, …);
```

- 실행 예시

```
DB_2025-12345> insert into account values(9732, 'Perryridge');
DB_2025-12345> 1 row inserted
```

[구현 요구사항]

- 올바른 INSERT문의 실행
  - ✓ 적절한 컬럼에 입력된 값을 삽입하고, `InsertResult` 메시지 출력.
  - ✓ `char` 컬럼 타입에 허용하는 최대 길이보다 긴 문자열을 삽입하려 할 경우, 길이에 맞게 자른 (truncate) 문자열을 삽입한다.
- 쿼리 오류에 대한 종류별 메시지 출력
  - ✓ 삽입할 테이블이 존재하지 않을 경우, `NoSuchTable(#commandName)` 메시지 출력.
  - ✓ 삽입할 데이터의 타입이 맞지 않는 경우, `InsertTypeMismatchError` 메시지 출력.
    - 지정된 컬럼과 값의 개수가 다른 경우
    - 지정된 컬럼과 값의 타입이 맞지 않는 경우
    - 컬럼을 명시하지 않았는데, 입력 값 개수와 해당 테이블의 attribute 수가 다른 경우
  - ✓ `null` 값을 가질 수 없는 컬럼에 `null`을 삽입하는 경우, `InsertColumnNonNullableError(#colName)` 메시지 출력.
  - ✓ 존재하지 않는 column에 값을 삽입하는 경우, `InsertColumnExistenceError(#colName)` 메시지 출력.

2.2.2. DELETE

- 정의

```
delete from table_name [where clause];
```

- 실행 예시

```
DB_2025-12345> delete from account where branch_name = 'Perryridge';
DB_2025-12345> 5 rows deleted
```

[DELETE문 구현 요구사항]

- 올바른 DELETE문의 실행

- ✓ 테이블에서 조건에 맞는 튜플을 삭제하고 `DeleteResult(#count)` 메시지를 출력한다.
- ✓ WHERE절이 있다면, WHERE절의 조건을 만족하는 튜플을 삭제한다.
- ✓ WHERE절이 없다면, 모든 튜플을 삭제한다.
- ✓ 다른 테이블에서 foreign key로 참조하고 있는 튜플이 삭제 대상에 포함될 경우 DeleteReferentialIntegrityPassed(#count) 메시지를 출력한다. 이 때 #count는 삭제 요청된 모든 튜플의 개수를 의미한다.
- ✓ Foreign key constraint를 위반하지 않고 삭제할 수 있는 나머지 튜플도 삭제하지 않는다.

- 쿼리 오류에 대한 종류별 메시지 출력
  - ✓ 삭제 요청한 테이블이 존재하지 않는 경우, `NoSuchTable(#commandName)` 메시지 출력.
    - WHERE절 내에서 발생한 오류 처리는 아래 WHERE절 구현 요구사항을 따른다.

[WHERE절 구현 요구사항]

- WHERE절에서는 아래 조건에 부합하는 경우에만 비교연산자로 두 값을 비교할 수 있다.

Data Type	동일 여부 (=, =)	대소 비교 (>, <, >=, <=)
char	√	
int, date	√	√

Table . 데이터 타입별 가용 비교연산자

- ✓ `char`, `int`, `date` 타입의 값은 각각 동일한 타입의 값과 Table 1의 연산자를 사용하여 비교할 수 있다.
- ✓ `null`은 다른 모든 타입의 값과 비교(is, is not)할 수 있다.

- WHERE절 내에서 발생한 오류 처리
  - ✓ `WHERE`절에서 비교 연산자로 비교할 수 없는 값들을 비교할 경우, `IncomparableError` 메시지 출력
  - ✓ `FROM`절에 명시되지 않은 테이블을 `WHERE`절에서 참조할 경우, `TableNotSpecified(#clauseName)` 메시지 출력
  - ✓ `WHERE`절에서 존재하지 않는 컬럼을 참조할 경우, `ColumnNotExist(#clauseName)` 메시지 출력
  - ✓ `WHERE`절에서 참조하는 컬럼이 어느 테이블에 속해 있는지 모호한 경우, `AmbiguousReference(#clauseName)` 메시지 출력

- [where clause]의 conditions는 2개 이하의 simple condition으로 구성되는 경우만 상정한다.
  - ✓ E.g.
    - (1 condition) where *condition1*
    - (2 conditions) where *condition1* and *condition2*
      where *condition1* or *condition2*

2.2.3. SELECT

- 정의

```
select [aggregate_func(][table_name.]column_name[)], …
from table_name
[join table_name on join_condition]
[where clause]
[order by column_name [asc|desc]];
```

- 실행 예시

```
DB_2025-12345> select * from account;
--------------------------------------
account_number | branch_name | balance
A-101          | Downtown    | 500
A-102          | Perryridge  | 400
A-201          | Brighton    | 900
A-215          | Mianus      | 700
A-217          | Brighton    | 750
A-222          | Redwood     | 700
A-305          | Round Hill  | 350
--------------------------------------
7 rows in set
DB_2025-12345> select customer_name, borrower.loan_number, amount
from borrower
join loan on borrower.loan_number = loan.loan_number
where branch_name = 'Perryridge';
order by customer_name asc;
-----------------------------------
customer_name | loan_number | amount
Adams         | L-16        | 1300
Hayes         | L-15        | 1500
-----------------------------------
2 rows in set
DB_2025-12345>
```

[구현 요구사항]

- 올바른 SELECT문의 실행
  - ✓ 아무 조건이 없다면, 선택한 테이블 내 모든 튜플에서 요청된 컬럼에 해당하는 값을 출력한다.
  - ✓ WHERE절이 있다면, 2.2.2. DELETE의 [WHERE절 구현 요구사항]을 따라 조건에 맞는 튜플에서 출력한다.
  - ✓ JOIN 구문이 있다면, [JOIN 구현 요구사항]에 따라 테이블을 조인하여 출력한다.
  - ✓ ORDER BY 구문이 있다면, [ORDER BY 구현 요구사항]에 따라 정렬하여 출력한다.
    - ○ ORDER BY 구문이 없다면 튜플의 출력 순서가 달라도 정답 처리한다.
  - ✓ 출력 포맷은 실행 예시를 따르되, 점선, 필드 간의 공백은 개수나 모양에 제약을 두지 않는다.
- 쿼리 오류에 대한 종류별 메시지 출력
  - ✓ WHERE절 내에서 발생한 오류 처리는 [WHERE 구현 요구사항]을 따른다.

- ✓ FROM절에 존재하지 않는 테이블이 포함되어 있다면, `SelectTableExistenceError(#tableName)` 메시지 출력.
- ✓ SELECT 대상이 되는 컬럼이 어느 테이블에 속한 것인지 모호하거나, 해당 컬럼이 없는 경우 `SelectColumnResolveError(#colName)` 메시지 출력.

- SELECT 구문의 FROM절은 **3개 이하**의 테이블을 포함하는 경우만을 가정한다.

[JOIN 구현 요구사항]

- INNER JOIN을 구현한다.
  - ✓ "INNER"는 항상 생략한다.
  - ✓ OUTER JOIN에 해당하는 부분은 고려하지 않는다.
- 3개 이하의 테이블을 조인하는 경우만을 상정한다.
- On
  - ✓ On의 조건절은 table1.column1 = table2.column2의 형태를 따르며 이외의 경우는 평가하지 않는다.
  - ✓ 같은 타입을 가지는 두 컬럼만 사용할 수 있다.
  - ✓ JOIN 대상을 가리킬 때 On을 사용하는 경우만 고려한다.
- 오류 처리
  - ✓ 존재하지 않는 컬럼을 사용한 경우, `ColumnNotExist(#clauseName)` 메시지 출력.
  - ✓ 서로 다른 타입의 두 컬럼을 사용한 경우, `IncomparableError` 메시지 출력.

[ORDER BY 구현 요구사항]

- 오름차순 정렬인 ASC와 내림차순 정렬인 DESC를 구현한다.
  - ✓ ASC 혹은 DESC을 사용한 ORDER BY절만을 평가한다.
- 오류 처리
  - ✓ ORDER BY절에서 존재하지 않는 컬럼을 참조한 경우, `ColumnNotExist(#clauseName)` 메시지 출력.
  - ✓ ORDER BY절에서 참조하는 컬럼이 어느 테이블에 속해 있는지 모호한 경우, `AmbiguousReference(#clauseName)` 메시지 출력.

2.3. (Optional) GROUP BY

본 장에서는 GROUP BY를 포함하는 SELECT 구문과 그 구현에 대해 설명한다. 이는 필수 구현 사항이 아니므로 **Q&A는 제공되지 않는다**.

- 정의

```
select [aggregate_func(][table_name.]column_name[)], …
from table_name
[join table_name on join_condition]
[where clause]
[group by column_name]
[order by column_name [asc|desc]];
```

- 예시

```
DB_2025-12345> select customer.name, max(account.balance)
from customer
join account on customer.name = account.customer_name
group by customer.name
order by customer.name desc;
-------------------------------------
customer.name  | max(account.balance)
Charles        | 1300
Betty          | 1200
Albert         | 1500
-------------------------------------
3 rows in set
DB_2025-12345>
```

[GROUP BY 구현 요구사항]

- GROUP BY에 명시된 컬럼을 기준으로 묶어 SELECT문에 명시된 컬럼과 Aggregate function의 결과에 해당하는 값을 출력한다.
- SELECT의 대상이 되는 컬럼은 반드시 GROUP BY절에 명시되거나 Aggregate function을 사용해야 한다.
- Aggregate function은 MAX, MIN, SUM을 구현한다.
  - ✓ MAX는 NULL이 아닌 값들 중 가장 큰 값을 반환한다.
    - STR 타입의 경우 사전식(lexicographical) 순서를 따르며 영문 문자열만을 평가한다.
    - DATE 타입의 경우 더 나중의 날짜를 더 큰 것으로 한다.
    - DB가 비어 있거나 NULL값만 존재하는 경우 NULL을 반환한다.
  - ✓ MIN은 NULL이 아닌 값들 중 가장 작은 값을 반환한다.
  - ✓ SUM은 NULL을 제외한 값의 합계를 반환한다.
    - INT 타입만 처리하고, 그 외의 경우에는 0을 반환한다.
    - DB가 비어 있거나 NULL값만 존재하는 경우 0을 반환한다.
- 한 개의 컬럼을 기준으로 하는 경우만을 평가한다.
- GROUP BY절 내에서 발생한 오류 처리

- ✓ FROM절에 명시되지 않은 테이블을 GROUP BY절에서 참조할 경우, `TableNotSpecified(#clauseName)` 메시지 출력.
- ✓ GROUP BY절 내에서 존재하지 않는 컬럼을 참조할 경우, `ColumnNotExist(#clauseName)` 메시지 출력.
- ✓ GROUP BY절에서 참조하는 컬럼이 어느 테이블에 속해 있는지 모호한 경우, `AmbiguousReference(#clauseName)` 메시지 출력.
- ✓ GROUP BY와 Aggregate function의 대상이 아닌 컬럼이 SELECT문에 존재할 경우, `SelectColumnNotGrouped(#colName)` 메시지 출력.

[ORDER BY 구현 요구사항]

- ORDER BY절에서는 Aggregate function을 사용하지 않는다.

3. 개발 환경

- Python 3.10 ~ 3.12
- Lark API
- Oracle Berkeley DB API

4. 제출

다음 파일들을 PRJ1-3_학번.zip(예: PRJ1-3_2025-12345.zip)으로 압축하여 제출한다. **압축 해제 시 DB 폴더 외에 폴더가 없어야 한다. 즉, 압축 해제 시 grammar.lark, run.py, 보고서 파일이 폴더 안에 들어 있지 않도록 압축하여 제출한다.**

1. grammar.lark
2. run.py
   - ✓ **프로젝트의 최상위 디렉토리에 위치해야 한다.**
   - ✓ 추가적인 소스코드 파일 및 서브 디렉토리를 함께 제출해도 된다. 단, python run.py로 프로그램이 구동될 수 있도록 해야 한다.
   - ✓ 적절한 주석을 포함해야 한다.
3. 리포트
   - ✓ **프로젝트의 최상위 디렉토리에 위치해야 한다.**
   - ✓ **반드시 pdf 형식이어야 한다.**
   - ✓ 파일명은 PRJ1-3_학번.pdf (예: PRJ1-3_2025-12345.pdf)으로 한다.
   - ✓ 2장 이내로 작성한다(1장을 권장).
   - ✓ 반드시 포함되어야 하는 내용
     - 핵심 모듈과 알고리즘에 대한 설명
     - 구현한 내용에 대한 간략한 설명
     - (제시된 요구사항 중 구현하지 못한 부분이 있다면) 구현하지 못한 내용
     - 프로젝트를 하면서 느낀 점 및 기타사항
   - ✓ 추가로 포함할 수 있는 내용
     - 정의되지 않은 요소의 구현
     - 본 문서에 정의된 오류 유형 외 추가로 정의한 오류 유형
4. (Optional) DB 폴더 및 파일
   - ✓ Database 저장 시 경로는 다음과 같다.
     - DB를 단일 파일로 구성할 경우:
       PRJ1-3_2025-12345/myDB.db
     - DB를 복수개의 파일로 구성할 경우:
       PRJ1-3_2025-12345/DB/myDB1.db ,
       PRJ1-3_2025-12345/DB/myDB2.db , …

```
# 제출 파일(PRJ1-3_학번.zip) 내부 구조
PRJ1-3_학번
```

```
├── PRJ1-3_학번.pdf
├── run.py
├── grammar.lark
└── myDB.db (DB 파일이 복수개로 저장되는 경우 DB/myDB1.db, …)
```

5. 성적 관련 사항

- 기한 끝난 후 과제 제출 시 늦은 시간별 감점 기준
  - ✓ 0~24 시간: 10% 감점
  - ✓ 24~48 시간: 20% 감점
  - ✓ 48시간 이후: 점수 없음
- 부정 행위는 0점 처리
  - ✓ 다른 사람의 코드를 참조하는 행위
  - ✓ 이전에 수강한 사람의 코드를 참조하는 행위
  - ✓ 프로그램을 통해 표절 여부 확인 예정
- 본 문서 상의 출력 양식을 지키지 않을 시 감점
- 소스 코드에 주석이 없는 경우 감점

6. 참고 자료

- **Oracle Berkeley DB**
  - ✓ http://www.oracle.com/technetwork/database/berkeleydb/overview/index.html
- **Python Berkeley DB API Resources**
  - ✓ https://www.jcea.es/programacion/pybsddb.htm
  - ✓ https://docs.jcea.es/berkeleydb/latest/index.html