[AI초급] SQLD 자격증 코스 - 챕터 8

매 주차 강의자료 시작에 PDF파일을 올려두었어요!

▼ PDF 파일

[수업 목표]

- SQL에 대해 알아보고 기본기를 학습합니다.

[목차]

모든 토글을 열고 닫는 단축키
Windows : `Ctrl` + `alt` + `t`
Mac : `⌘` + `⌥` + `t`

---

01. DML

SQL의 한 종류인 DML에 대해 학습합니다

▼ 1) INSERT

☑️ DML의 정의

DML(Data Manipulation Language)은 테이블에 데이터를 삽입(INSERT), 삭제(DELETE), 수정(UPDATE), 조회(SELECT) 하는 일을 수행하는 SQL로 구성되어 있습니다. 추가적으로 데이터를 삽입, 삭제, 수정하는 SQL문을 수행하고 난 이후에 이를 데이터베이스에 완전하게 반영하는 커밋(COMMIT)과 데이터베이스 작업의 취소를 위한 롤백(ROLLBACK)이 있습니다.

☑️ INSERT

테이블이 준비 되었다면 필요한 데이터를 저장을 해야 합니다. 이때 사용하는 명령어가 바로 `INSERT` 문입니다. 데이터를 저장할 때는 2가지 방법이 있습니다.

1. 칼럼 리스트를 명시하는 방법입니다.
    - 어떤 칼럼에 값을 넣을지 명시하며 저장할 데이터도 칼럼의 순서에 맞게 작성합니다.
2. 칼럼 리스트를 명시하지 않는 방법입니다.
    - 칼럼 리스트를 명시하지 않으면 모든 칼럼에 값을 넣을 것이라는 의미가 됩니다.
    - 테이블에서 정의된 칼럼의 개수에 맞추어 저장되는 값을 순서대로 입력하면 됩니다.

- 구조

```
-- 칼럼 리스트 명시

INSERT INTO 테이블_이름 (컬럼1, 컬럼2, ...) VALUES (값1, 값2, ...);
```

```
-- 칼럼 리스트 명시 x

INSERT INTO 테이블_이름 VALUES (값1, 값2, ...);
```

- 예시
  - player 테이블에 이름이 KIM, 키는 184, 몸무게는 75 선수 데이터 추가하기

```
-- 칼럼 리스트 명시

INSERT INTO player (name, height, weight) VALUES ('KIM', 184, 75);

-- 칼럼 리스트 명시 x

INSERT INTO player VALUES ('LEE', 187, 74);
```

- 실습

> **pokemon 테이블에**
> **포켓몬 번호 25번 이름 pikachu 속성은 Electric**
> **포켓몬 번호 6번 이름 charizard 속성은 Fire를 저장하세요.**

```
-- 테이블 구조

CREATE TABLE pokemon (
  pm_id INT PRIMARY KEY NOT NULL,
  name VARCHAR(20) NOT NULL,
  attr VARCHAR(20) DEFAULT 'normal'
);
```

```
-- Oracle

INSERT INTO pokemon (pm_id, name, attr) VALUES (25, 'pikachu', 'Electric');
INSERT INTO pokemon (pm_id, name, attr) VALUES (6, 'charizard', 'Fire');
```

▼ 2) UPDATE

한 번 입력된 데이터를 수정해야 할 때, `UPDATE` 문을 사용하여 내용을 수정할 수 있습니다. 이때 `WHERE` 문을 사용하여 조건에 맞는 데이터를 찾아 해당 데이터를 수정할 수 있습니다.

- 구조

```
UPDATE 테이블_이름 SET 수정_할_컬럼_이름 = 수정_할_새로운_값, ...;
```

- 예시
  - player 테이블의 back_no를 99로 변경하기
  - player 테이블의 position을 MF로 변경하기

```
UPDATE player SET back_no = 99;

UPDATE player SET position = 'MF';
```

▼ 아하 - 그렇구나 하고 넘기면 말이죠 👀

꼬마 개발자 시절 **where** 없이 **update** 날린적 있습니다.

---

f5를 누르자 마자 됐다는걸 깨닳았는데

ms sql이라 commit고 rollback이고 할것 없이

이미 데이터는 저세상 데이터가 되었습니다

같이 화면을 보고 있던 영업사원이 대형 사고가

벌어졌다는걸 눈치채고 빠르게 도망가더군요.....

400만건 정도의 데이터 날라가고 어찌어찌 복구는 했습니다만

아직도 쿼리문 작성할때 where 구절 먼저 작성합니다...

F5 눌렀을때 정수리에서부터 등뼈를 타고 내리는 그 소름끼치는 기분이란 ㄷㄷㄷㄷ

모두의 공원 델리게이트님 작성글 발췌

→ 지금은 하하하 ! 하고 넘어가겠지만 정작 한 번 겪어보면 😇

▼ 3) DELETE

데이터를 삭제하기 위한 명령어 `DELETE`입니다. 삭제할 때 `WHERE` 을 이용하여 조건을 만족하는 데이터만 선택하여 삭제할 수 있습니다.

- 구조

```
DELETE [FROM] 테이블_이름;

DELETE [FROM] 테이블_이름 WHERE 삭제할_조건;
```

- 예시
  - player 테이블의 모든 데이터를 삭제하기
  - player 테이블에서 back_no가 50인 데이터만 삭제하기

```
DELETE FROM player;

DELETE FROM player WHERE back_no = 50;
```

🆚 DELETE vs TRUNCATE

`DELETE` 명령어는 실행하면 바로 데이터가 삭제되지 않습니다. '`COMMIT`'이라는 명령어가 실행되고 난 이후에 DB에 최종 적용됩니다. 앞서 배운 DDL의 경우 바로 실제 테이블에 반영이 되었지만, DML 명령어들은 데이터가 처리될 때 실패하거나 오류가 발생할 수 있기 때문에 명령어를 입력하는 순간 실제 테이블이 완벽히 반영되지 않습니다. 만약 데이터를 `DELETE`를 이용하여 잘못 삭제했다면 `ROLLBACK` 명령어를 통해 다시 되돌릴 수 있습니다. (단, SQL Server의 경우는 DML의 경우도 AUTO COMMIT 이 되기 때문에 따로 COMMIT 명령어를 입력하지 않아도 됩니다) COMMIT과 ROLLBACK은 다음 파트에서 다룹니다.

> 💡 ❕ **참고 - DDL의 TRUNCATE 와 삭제 동작 비교**
>
> - DELETE 명령어는 데이터가 삭제될 때 복원을 위해 log를 남기기 때문에 시스템에서 데이터 log를 기록하고 저장하기 위한 작업으로 인한 약간의 부하가 있습니다.
> - DDL의 TRUNCATE와 데이터 삭제라는 유사한 동작을 진행하지만 가장 큰 차이점은 DDL 명령어는 바로 테이블에 적용이 되기 때문에 ROLLBACK이 불가능합니다. 즉, TRUNCATE로 삭제된 데이터는 되돌릴 수 없습니다.
> - 다만, 복원을 위한 log를 남기지 않기 때문에 TRUNCATE로 모든 데이터를 지울 때 시스템의 부하가 적다는 것이 특징입니다.

```
-- Oracle

DELETE FROM player;          -- player 테이블에서 데이터 삭제
ROLLBACK;                    -- 되돌리기
SELECT COUNT(*) FROM player; -- player 테이블의 전체 데이터 개수 구하기
-- 480

TRUNCATE TABLE player;       -- player 테이블에서 데이터 삭제
ROLLBACK;                    -- 되돌리기
SELECT COUNT(*) player;      -- player 테이블의 전체 데이터 개수 구하기
-- 0
```

- 실습

```
CREATE TABLE pokemon (
  pm_id INT PRIMARY KEY NOT NULL,
  name VARCHAR(20) NOT NULL,
  attr VARCHAR(20) DEFAULT 'normal'
);

INSERT INTO pokemon VALUES (1, 'Bulbasaur', 'grass');
INSERT INTO pokemon VALUES (2, 'Ivysaur', 'grass');
INSERT INTO pokemon VALUES (3, 'Venusaur', 'grass');
```

데이터 삭제는 이렇게 하세요

- 데이터 '삭제'는 굉장히 비지니스에 큰 영향을 끼칠 수 있는 작업입니다.
- 잘못삭제한 데이터는 되돌릴 수 없는 경우도 많고, 전체를 되돌리는 것이 불가능한 경우가 대부분입니다. 여러분의 저녁이 아니라 주말을 없앨 수 있는 마법같은 작업입니다.
- 따라서 `DELETE` 를 하기전에는 반드시 조회, 즉 `SELECT` 를 해보고 내가 지우려는 데이터가 맞는지 확인한 뒤 지우는 습관을 들이는 것이 좋습니다. `SELECT` 만 바로 `DELETE` 로 바꿔주면 되기때문에 번거롭지도 않아요 🙂

  (이렇게 알려드려도 결국 한 번은 고생을 해봐야 적용하시긴 하더라구요)

1. pokemon 테이블에서 pm_id 가 2인 데이터만 삭제하세요.

```
-- Oracle

-- 데이터 확인하기
-- SELECT * FROM pokemon WHERE pm_id = 2;
```

```
-- 데이터 삭제하기
DELETE FROM pokemon WHERE pm_id = 2;
```

2. DELETE와 TRUNCATE의 차이를 pokemon 테이블을 삭제하는 과정을 통해 학습해보세요.

```
-- DELETE

DELETE FROM pokemon;
ROLLBACK;
SELECT COUNT(*) FROM pokemon;
-- 2 (위에서 삭제하고 남은 2개의 row)
```

```
-- TRUNCATE

TRUNCATE TABLE pokemon;
ROLLBACK;
SELECT COUNT(*) FROM pokemon;
-- 0 (데이터 모두 삭제)
```

DROP	TRUNCATE	DELETE
DDL	DDL (일부 DML 느낌)	DML
ROLLBACK 불가능	ROLLBACK 불가능	COMMIT 이전 ROLLBACK 가능
AUTO COMMIT	AUTO COMMIT	사용자 COMMIT
테이블의 정의 자체를 완전히 삭제	테이블을 최초 생성된 초기 상태로 만듦	테이블은 그대로 두고 데이터만 삭제

▼ 연습문제

☑️ 문제 1

Q. 문제	다음 중 ROLLBACK을 통해 작업을 되돌릴 수 있는 것은?
A. (1)	CRAETE
A. (2)	TRUNCATE
A. (3)	DROP
A. (4)	DELETE

▼ 정답

(4) COMMIT 단위로 되돌릴 수 있다.

☑️ 문제 2 (주관식)

Q. 문제	pokemon 테이블에서 ‘zoozoo’ name을 갖는 데이터를 삭제하는 SQL을 작성해보세요.
A.	(SQL을 작성하세요)

▼ 정답

`DELETE FROM pokemon WHERE name=’zoozoo’;`

☑️ 문제 3

Q. 문제	(a), (b)에 들어갈 단어를 옳게 짝지은 것은? 데이터를 삭제하는 방법에는 크게 (a)를 사용하는 방법과 (b)를 사용하는 방법이 있다. (a)는 DDL 명령어로 테이블을 생성한 초기상태로 만들며, ROLLBACK이 불가능하다. (b)는 DML 명령어로 해당 하는 데이터만 삭제한다.

	COMMIT을 통해 반영하고, 이 COMMIT 단위로 ROLLBACK이 가능하다.
A. (1)	(a)DELETE, (b)DROP
A. (2)	(a)DELETE, (b)TRUNCATE
A. (3)	(a)UPDATE, (b)DELETE
A. (4)	(a)TRUNCATE, (b)DELETE

▼ 정답

(4)

02. SELECT

✔️ **가장 많이 사용하게 될 SELECT문과 연산자에 대해 학습합니다**

▼ 1) SELECT

SELECT는 테이블에 데이터를 조회할 때 사용하며, SQL에서 가장 많이 사용하는 구문입니다. `SELECT`문은 다양한 옵션이 함께 적용되어 사용되기 때문에 많은 활용이 있는데, 여기서는 먼저 `SELECT`의 기본 형태에 대해서 알아보겠습니다.

- 구조

  SELECT에서 원하는 데이터를 보기 위해서는 보고싶은 컬럼과 데이터를 가져올 테이블을 적어주면 됩니다.

```
-- person 테이블에서 모든 name 데이터 가져오기

SELECT name FROM person;
```

  여러개의 컬럼을 한 번에 가져올수도 있습니다.

```
-- person 테이블에서 모든 name, age 데이터 가져오기

SELECT name, age FROM person;
```

  만약 모든 칼럼을 다 가져오고 싶다면 `*` 를 이용하여 작성할 수 있습니다.

```
-- 모든 컬럼의 데이터 가져오기

SELECT * FROM 테이블_이름;
```

ALIAS

만약 칼럼의 명이 너무 길거나 더 구분하기 쉬운 다른 명칭으로 조회하고 싶을 때는 `ALIAS`를 활용할 수 있습니다. alias는 별명이라는 뜻으로 특정 칼럼이나 테이블의 이름을 바꿔서 조회할 수 있습니다. 주로 쿼리 결과의 가독성을 높이거나 복잡한 테이블명이나 컬럼명을 간략하게 표현할 때 사용됩니다

- **컬럼 별칭 사용하기**

  컬럼의 의미를 더 명확하게 표현하거나, 계산된 컬럼의 결과를 더 직관적으로 표현하는데 사용됩니다. 이때, 테이블의 칼럼 이름이 직접 변경되는 것이 아닌 칼럼 명을 일회성으로 수정하여 결과에만 나타낸다는 것에 주의하세요! 칼럼명 바로 뒤에 `AS`라는 키워드를 사용하여 나타낼 수 있으며 만약 칼럼 명에 띄어쓰기나 특수문자를 포함, 혹은 대/소문자를 구분할 필요가 있을 때는 따옴표`'` 를 이용하여 작성합니다.

```
-- 특정 컬럼을 별명으로 바꿔서 데이터 가져오기

SELECT 컬럼_이름 AS '별명' FROM 테이블_이름;
```

```
SELECT first_name, last_name, salary*12 AS annual_salary
FROM employees;

-- 이때 AS 키워드를 생략할 수 있습니다
SELECT first_name, last_name, salary*12 annual_salary
FROM employees;
```

```
+------------+-----------+---------------+
| first_name | last_name | annual_salary |
+------------+-----------+---------------+
| John       | Doe       | 60000         |
| Jane       | Smith     | 72000         |
| Michael    | Johnson   | 90000         |
+------------+-----------+---------------+
```

- **테이블 별칭 사용하기**

  테이블 별칭은 쿼리 내에서 테이블 이름에 대한 대체 이름을 지정하는 것으로 여러 테이블을 하나의 SQL문에서 조회할 때 많이 사용됩니다. 이부분에 대해서는 뒤에 조인(JOIN)을 다룰때 조금 더 자세히 살펴보고 여기서는 간단히 테이블 별칭에 대해서 알아봅니다.

  테이블 별칭은 `AS` 키워드 없이 작성합니다.

  ```
  SELECT e.first_name, e.last_name FROM employees e
  ```

- ALIAS 부여 정리

유형	부여 방식	설명
칼럼	칼럼명 AS ALIAS(앨리어스)명	칼럼명과 앨리어스명 사이에 AS을 써도 되고 안 써도 됨
테이블	테이블명 ALIAS(앨리어스)명	테이블과 앨리어스명 사이에 AS 사용할 수 없음

DISTINCT

`DISTINCT`는 쿼리 결과에서 중복된 값을 제거하여 고유한 값만을 반환하도록 하는 역할을 합니다. 이를 통해 특정 컬럼 또는 컬럼들의 조합에 대한 중복을 제거하여 결과를 보다 정제된 형태로 가져올 수 있습니다.

예를들어, 아래와 같은 `employees` 테이블이 있다면

```
+------------+-----------+
| first_name | last_name |
+------------+-----------+
| John       | Doe       |
| Jane       | Smith     |
| John       | Doe       |
| Michael    | Johnson   |
+------------+-----------+
```

여기에서 중복 없이 `first_name` 을 조회하고 싶을때 아래와 같이 작성할 수 있습니다.

```
SELECT DISTINCT first_name FROM employees;
```

```
-- 결과
+------------+
| first_name |
+------------+
| John       |
| Jane       |
| Michael    |
+------------+
```

그럼 만약 `DISTINCT`와 함께 컬럼을 여러개 적게되면 어떻게 될까요?

```
CREATE TABLE employees (
    employee_id NUMBER,
    first_name VARCHAR2(50),
    last_name VARCHAR2(50),
    department_id NUMBER
);

INSERT INTO employees VALUES (1, 'John', 'Doe', 101);
INSERT INTO employees VALUES (2, 'Jane', 'Smith', 102);
INSERT INTO employees VALUES (3, 'John', 'Doe', 101);
INSERT INTO employees VALUES (4, 'Michael', 'Johnson', 103);
```

직접 실습해보고 싶다면 위 명령어를 이용하세요🙂

```
+------------+-----------+
| first_name | last_name |
+------------+-----------+
| John       | Doe       |
| Jane       | Smith     |
| John       | Doe       |
| Michael    | Johnson   |
+------------+-----------+
```

이러한 데이터가 있는 테이블에서 아래와 같은 쿼리를 하게되면

지정한 컬럼들의 값의 조합이 고유한 경우에만 결과에 포함됩니다.

```
SELECT DISTINCT first_name, last_name FROM employees;
```

```
FIRST_NAME   LAST_NAME
----------   ---------
John         Doe
Jane         Smith
Michael      Johnson
```

- 추가 예시
  - **ALIAS**
    - player_name 이라는 칼럼 명을 name이라는 별명으로, back_no 의 별명은 '등 번호'로 설정하여 player 테이블 조회하기
    - player 테이블의 모든 칼럼을 조회하기

    ```
    SELECT player_name AS name, back_no '등 번호' FROM player;

    SELECT * FROM player;
    ```

  - **DISTINCT**
    - position을 중복 없이 조회하기

```
SELECT DISTINCT position FROM player;
```

- 실습

```
CREATE TABLE pokemon (
  pm_id INT PRIMARY KEY NOT NULL,
  name VARCHAR(20) NOT NULL,
  attr VARCHAR(20) DEFAULT 'normal'
);

INSERT INTO pokemon VALUES (1, 'Bulbasaur', 'grass');
INSERT INTO pokemon VALUES (2, 'Ivysaur', 'grass');
INSERT INTO pokemon VALUES (3, 'Venusaur', 'grass');
INSERT INTO pokemon VALUES (4, 'Charmander', 'Fire');
INSERT INTO pokemon VALUES (5, 'Charmeleon', 'Fire');
INSERT INTO pokemon VALUES (25, 'Pikachu', 'Electric');
```

1. pokemon 테이블의 칼럼명을 번호, 이름, 속성 이라는 한글 별명으로 설정하여 데이터를 조회하세요.

```
-- Oracle

SELECT pm_id AS 번호, name AS 이름, attr AS 속성 FROM pokemon;

-- 아래처럼 조회 했을 때의 결과와 비교
SELECT * FROM pokemon;
```

```
-- SELECT pm_id AS 번호, name AS 이름, attr AS 속성 FROM pokemon;
+------+-----------+---------+
| 번호  | 이름       |  속성    |
+------+-----------+---------+
| 1    | Bulbasaur | grass   |
| 2    | Ivysaur   | grass   |
| 3    | Venusaur  | grass   |
| 4    | Charmander| Fire    |
| 5    | Charmeleon| Fire    |
| 25   | Pikachu   | Electric|
+------+-----------+---------+

-- SELECT * FROM pokemon;
+------+-----------+---------+
| pm_id| name      | attr    |
+------+-----------+---------+
| 1    | Bulbasaur | grass   |
| 2    | Ivysaur   | grass   |
| 3    | Venusaur  | grass   |
| 4    | Charmander| Fire    |
| 5    | Charmeleon| Fire    |
| 25   | Pikachu   | Electric|
+------+-----------+---------+
```

2. pokemon 테이블에 저장된 속성을 중복 값 없이 조회하세요.

```
-- Oracle

SELECT DISTINCT attr FROM pokemon;
```

```
-- SELECT DISTINCT attr FROM pokemon;
+---------+
| attr    |
+---------+
| grass   |
| Fire    |
```

```
| Electric|
+---------+
```

▼ 2) 산술/합성 연산자

☑️ 산술 연산자

SQL에서 사용하는 산술 연산자라고 특별한 형태를 가진 것은 아닙니다. 이미 익숙하게 사용하고 있는 `+`, `-`, `*`, `/` 와 같은 산술 연산자와 우선순위를 지정하는 `( )` 를 연산에 활용할 수 있습니다. 숫자 데이터 타입의 값을 다룰 때 사용되며, 기본적인 수학 연산을 수행합니다.

- `+` : 더하기 연산
- `-` : 빼기 연산
- `*` : 곱하기 연산
- `/` : 나누기 연산
- `%` : 나머지 연산
- 연산자 우선순위

  가장 높음 `( )` → `*` → `/` → `+` → `-` 가장 낮음

위 연산자를 SELECT 문에 적용하여 연산된 데이터를 조회할 수 있습니다. 산술 연산은 기본적으로 숫자 데이터에 대해 사용할 수 있으며 저장된 데이터형이 INT(NUMBER)이거나, 문자이지만 숫자 형태로 구성된 경우 산술 연산을 할 수 있습니다.

- 구조

```
SELECT 숫자형1 - 숫자형2 FROM 테이블_이름;
```

- 예시
  - 사칙연산 예시

```
-- 아래의 결과는 어떻게 될까요?

SELECT 10 + 5 AS addition,
       15 - 7 AS subtraction,
       6 * 3 AS multiplication,
       20 / 4 AS division,
       17 % 5 AS modulo
FROM dual;
```

```
+----------+------------+----------------+----------+--------+
| ADDITION | SUBTRACTION| MULTIPLICATION| DIVISION | MODULO |
+----------+------------+----------------+----------+--------+
| 15       | 8          | 18             | 5        | 2      |
+----------+------------+----------------+----------+--------+
```

- 실습

> pocketmon 테이블에서 Pikachu의 몸무게는 현재 kg 단위로 저장이 되어 있습니다. SELECT를 이용하여 kg 기준이 아닌 파운드 단위(lbs) 기준으로 변환된 값을 조회해보세요. (1kg = 2.2 lbs)

```
CREATE TABLE pokemon (
  pm_id INT PRIMARY KEY NOT NULL,
  name VARCHAR(20) NOT NULL,
  attr VARCHAR(20) DEFAULT 'normal',
  weight INT
);

INSERT INTO pokemon VALUES (1, 'Bulbasaur', 'grass', 30);
INSERT INTO pokemon VALUES (2, 'Ivysaur', 'grass', 50);
INSERT INTO pokemon VALUES (3, 'Venusaur', 'grass', 150);
INSERT INTO pokemon VALUES (4, 'Charmander', 'Fire', 80);
```

```
INSERT INTO pokemon VALUES (5, 'Charmeleon', 'Fire', 200);
INSERT INTO pokemon VALUES (25, 'Pikachu', 'Electric', 15);
```

```
-- Oracle

SELECT weight * 2.2 FROM pokemon;
```

☑️ 합성 연산자

- 구조

  합성 연산자는 서로 다른 두 문자를 하나의 문자열로 합칠 때 사용합니다. 이때 DBMS별로 서로 다른 기호를 사용합니다. Oracle의 경우는 `||` 를 이용하고 SQL Server의 경우는 `+` 기호를 이용하여 서로 다른 두 문자를 합치게 됩니다.

```
-- Oracle

SELECT 문자형1 || 문자형2 FROM 테이블_이름;
```

```
-- SQL Server

SELECT 문자형1 + 문자형2 FROM 테이블_이름;
```

`CONCAT` 이라는 함수를 통해서도 두 문자를 하나의 문자열로 만들 수 있습니다.

```
SELECT CONCAT(문자형1, 문자형2) FROM 테이블_이름;
```

  이렇게 문자와 문자를 합치는 경우는 기존 정의된 칼럼이 아닌 결과에만 임시로 새로운 칼럼을 만들어서 합쳐진 문자열을 나타냅니다.

- 예시
  - player 테이블의 조회 결과를 'ㅇㅇㅇ선수, ㅇㅇㅇcm, ㅇㅇkg ' 으로 출력하기

```
-- Oracle

SELECT player_name || '선수,' || height || 'cm,' || weight || 'kg'
FROM player;
```

```
-- SQL Server

  SELECT player_name +'선수, '+ height +'cm, '+ weight +'kg' FROM player;
```

- 실습

> **pokemon 테이블에 저장된 키는 현재 cm 단위로 입력이 되어 있습니다.**
> **모든 데이터의 결과를 예시와 같이 조회하세요.**

번호	이름	속성	키
포켓몬번호: 1	이름: Bulbasaur	속성: grass	키: 50
포켓몬번호: 2	이름: Ivysaur	속성: grass	키: 90
포켓몬번호: 3	이름: Venusaur	속성: grass	키: 250
포켓몬번호: 4	이름: Charmander	속성: Fire	키: 80
포켓몬번호: 5	이름: Charmeleon	속성: Fire	키: 120
포켓몬번호: 25	이름: Pikachu	속성: Electric	키: 55

```
CREATE TABLE pokemon (
  pm_id NUMBER PRIMARY KEY NOT NULL,
  name VARCHAR2(20) NOT NULL,
  attr VARCHAR2(20) DEFAULT 'normal',
  height NUMBER
);

INSERT INTO pokemon VALUES (1, 'Bulbasaur', 'grass', 50);
INSERT INTO pokemon VALUES (2, 'Ivysaur', 'grass', 90);
INSERT INTO pokemon VALUES (3, 'Venusaur', 'grass', 250);
INSERT INTO pokemon VALUES (4, 'Charmander', 'Fire', 80);
INSERT INTO pokemon VALUES (5, 'Charmeleon', 'Fire', 120);
INSERT INTO pokemon VALUES (25, 'Pikachu', 'Electric', 55);
```

```
-- Oracle

SELECT
    '포켓몬번호: ' || pm_id 번호,
    '이름: ' || name 이름,
    '속성: ' || attr 속성,
    '키: ' || height 키
FROM pokemon;
```

▼ 연습문제

☑️ 문제 1

Q. 문제	아래의 `employees` 테이블에서 모든 직원의 이름과 입사한 해를 조회하는 SQL 문을 작성하세요. employees 테이블 구조: - employee_id (사원 번호) - first_name (이름) - last_name (성) - hire_year (입사 연도)
A. (1)	SELECT first_name, last_name FROM employees;
A. (2)	SELECT first_name, last_name, hire_year FROM employees;
A. (3)	SELECT employee_id, first_name, last_name FROM employees;
A. (4)	SELECT first_name, last_name, start_year FROM employees;

▼ 정답

(2)

☑️ 문제 2

Q. 문제	테이블 "Students"에서 모든 학생들의 이름과 수학 성적을 조회하려고 합니다. 수학 성적을 10점씩 더한 값으로 나타내려고 합니다. 올바른 SQL문을 고르세요.
A. (1)	SELECT name, math_score + 10 FROM Students;
A. (2)	SELECT name, math_score AS 'Math Score + 10' FROM Students;
A. (3)	SELECT name, math_score * 10 FROM Students;

A. (4)	SELECT name, math_score + 5 FROM Students;

▼ 정답

(1)

☑️ 문제 3

Q. 문제	테이블 "Orders"에서 고객들이 주문한 모든 도시를 나열하려고 합니다. 중복된 도시명은 제외하고 유일한 도시명만 조회하고 싶을때 올바른 SQL문을 고르세요.
A. (1)	SELECT UNIQUE city FROM Orders;
A. (2)	SELECT DISTINCT city FROM Orders;
A. (3)	SELECT city FROM Orders GROUP BY city;
A. (4)	SELECT city FROM DISTINCT Orders;

▼ 정답

(2)