Chapter 3-3: 데이터베이스를 똑똑하게 만들기 (중복 방지 로직)

Prologue: 같은 손님을 두 번 받지 않는 문지기

지금까지 우리 시스템은 어떤 데이터든 오는 대로 Notion DB에 들여보내는, 아주 친절하지만 멍청한 문지기와 같았습니다. 이제 우리는 이 문지기에게 '고객 명단'을 들려주고, "명단에 이미 있는 손님(데이터)이라면 정중히 돌려보내라"고 가르칠 것입니다.

이번 챕터에서는 Notion API의 `query` 기능을 사용하여, 데이터를 저장하기 전에 DB에 이미 존재하는지 먼저 확인하는 '중복 방지 로직'을 구현합니다. 이 미션을 완료하면, 신호용 님의 시스템은 비로소 **데이터의 무결성을 스스로 유지할 수 있는, 진정한 '데이터 파이프라인'**을 운영하는 개발자가 되는 것입니다.

1. 학습 목표 (Objectives)

- 데이터베이스에 데이터를 `create` 하기 전, `query`를 통해 먼저 조회하는 로직을 이해한다.
- `notion-client` 라이브러리의 `filter` 기능을 사용하여 특정 조건의 데이터만 검색할 수 있다.
- 데이터의 고유 식별자(Unique Identifier)의 중요성을 이해하고 설계에 적용할 수 있다.

2. 핵심 개념 (Core Concepts)

2.1 고유 식별자 (Unique Identifier): 데이터의 주민등록번호

중복을 확인하려면 "두 데이터가 같은지" 판단할 기준이 필요합니다. 공고 제목이나 회사명은 동명이인처럼 겹칠 수 있습니다. 하지만 **각 채용 공고의 상세 페이지 '링크(URL)'**는 절대로 중복되지 않는, 완벽한 '주민등록번호' 역할을 할 수 있습니다. 우리는 이 '링크'를 기준으로 중복 여부를 검사할 것입니다.

2.2 Notion API: `databases.query`

`notion.pages.create()`가 Notion에 데이터를 '쓰는' 기능이라면, `notion.databases.query()`는 데이터를 '읽는' 기능입니다. 우리는 이 `query` 기능에 `filter`라는 조건을 붙여, "이 데이터베이스에서, '링크' 속성이 우리가 가진 URL과 똑같은 데이터가 있는지 찾아봐줘" 라고 요청할 것입니다.

[Mission 11] 중복 공고 저장 방지 로직 구현

`main.py`의 Notion 저장 부분을 업그레이드하여, 똑똑한 문지기를 만들어봅시다.

> **Step 1: `main.py`의 Notion 저장 로직 수정**
>
> `main.py`의 `for` 루프 부분을 아래의 새로운 코드로 교체해주세요.

```
# main.py (중복 방지 로직이 추가된 최종 버전)

# ... (import 및 크롤러 실행 부분은 동일) ...

# 3. Notion에 저장
```

```
print(f"총 {len(all_jobs)}개의 채용 공고를 찾았습니다. Notion DB와 비교를 시작합니다...")

collection_date = datetime.now().strftime("%Y-%m-%d")
success_count = 0
duplicate_count = 0

for i, job in enumerate(all_jobs):
    link = job['link']
    title = job['title']

    # 1. '링크'를 기준으로 DB에 이미 데이터가 있는지 쿼리(조회)합니다.
    try:
        response = notion.databases.query(
            database_id=NOTION_DATABASE_ID,
            filter={"property": "링크", "url": {"equals": link}}
        )

        # 2. 쿼리 결과(response)에 데이터가 있는지 확인합니다.
        if len(response['results']) > 0:
            # 결과가 1개 이상 있다면, 이미 존재하는 데이터입니다.
            print(f"  [{i+1}/{len(all_jobs)}] [중복] {title}")
            duplicate_count += 1
            continue # 다음 루프로 바로 넘어갑니다.

    except Exception as e:
        print(f"  🚨 [오류 발생] Notion DB 조회 중 문제 발생: {e}")
        continue

    # 3. 중복이 아닐 경우에만 (위의 if문을 통과한 경우에만) 아래 코드가 실행됩니다.
    print(f"  [{i+1}/{len(all_jobs)}] [신규] {title} -> Notion에 저장 시도")
    company = job['company']
    source = job['source']

    try:
        notion.pages.create(
            parent={"database_id": NOTION_DATABASE_ID},
            properties={
                "직무": {"title": [{"text": {"content": title}}]},
                "회사명": {"rich_text": [{"text": {"content": company}}]},
                "링크": {"url": link},
                "수집일": {"date": {"start": collection_date}},
                "출처": {"rich_text": [{"text": {"content": source}}]}
            }
        )
        success_count += 1

    except Exception as e:
        print(f"  🚨 [오류 발생] '{title}' 저장 실패! 원인: {e}")

# 최종 결과 요약
print("-" * 20)
print(f"총 {len(all_jobs)}개의 공고 중,")
```

```
    print(f"  - {success_count}개의 새로운 공고를 Notion에 저장했습니다.")
    print(f"  - {duplicate_count}개의 공고는 이미 존재하여 건너뛰었습니다.")
    print("모든 작업이 완료되었습니다.")
```

다음 행동 계획 (Next Action)

1. ☐ `main.py`를 위 코드로 업데이트합니다.
2. ☐ `python main.py`를 실행합니다.
3. ☐ **첫 실행:** Notion DB가 비어있다면, 이전처럼 모든 공고가 `[신규]`로 표시되며 저장될 것입니다.
4. ☐ **두 번째 실행:** 코드를 전혀 바꾸지 않고 `python main.py`를 **바로 다시 한번 실행**합니다.
5. ☐ **결과 확인:** 이번에는 모든 공고가 `[중복]`으로 표시되고, "0개의 새로운 공고를 Notion에 저장했습니다." 라는 메시지가 뜨는지 확인합니다.

이 테스트까지 통과하면, 신호용 님의 시스템은 마침내 **지속적으로 실행해도 데이터베이스를 오염시키지 않는, 안정적인 자동화 시스템**으로 거듭나게 됩니다. 결과를 기다리겠습니다.