Chapter 4-6 (최종판): 데이터 파이프라인 완전 가동 및 고도화

Prologue: 모든 강물을 하나의 바다로, 그리고 보석을 캐내다

이제 우리는 흩어져 있던 모든 강물(3개 사이트)을 하나의 바다(Notion DB)로 모으는 대장정의 마지막 단계에 들어섭니다. 하지만 단순히 물만 모으는 것이 아닙니다. 신호용 님의 정밀한 분석 덕분에, 우리는 이제 강물 속에서 '신입'이라는 금맥을 찾아내고, '마감일'이라는 귀한 보석까지 함께 캐낼 수 있게 되었습니다.

이번 최종 챕터에서는, **모든 크롤러의 데이터 수집 기능을 완성**하고, **신입 필터와 마감일 수집이라는 고도화 기능까지 한번에 구현**하여, 이번 주 목표인 "완벽하게 쓸모 있는 자동화 시스템"을 완성할 것입니다.

1. 학습 목표 (Objectives)

- 지금까지 배운 모든 분석 및 코딩 기술을 총동원하여, `JobKoreaCrawler`와 `SaraminCrawler`의 모든 기능을 완성할 수 있다.
- URL 파라미터를 동적으로 제어하여, '신입' 공고만 필터링하는 사전 필터링 기능을 구현할 수 있다.
- 상세 페이지의 다양한 위치에 있는 특정 데이터('마감일')를 정확히 찾아내어 추출할 수 있다.
- Notion DB 스키마를 확장하고, 수집된 모든 데이터를 최종적으로 저장하는 완전한 파이프라인을 구축한다.

[Mission 25] 모든 크롤러 최종 완성

> **Step 1: `base_crawler.py`의 '최종 설계도' 완성**
>
> '신입' 필터링을 위해 `crawl` 메서드에 `is_newbie: bool` 파라미터를 추가합니다.

```
# crawlers/base_crawler.py (수정)

# ...
    @abstractmethod
    def crawl(self, keyword: str, pages_to_crawl: int = 1, is_newbie: bool =
False):
        # is_newbie: True일 경우 신입 공고만 필터링하도록 설계
        pass

    @abstractmethod
    def get_job_description(self, url: str) -> Dict[str, str]:
        # 반환 타입을 본문과 마감일을 모두 담을 수 있는 Dict로 변경
        """
        상세 페이지에서 '본문'과 '마감일'을 추출하여 딕셔너리로 반환합니다.
        e.g. {'description': '...', 'deadline': '2025-12-31'}
        """
        pass
# ...
```

Step 2: `jobkorea_crawler.py` **최종 구현**

신호용 님의 분석을 100% 반영하여 코드를 완성합니다.

```
# crawlers/jobkorea_crawler.py (최종 버전)

# ...
    def crawl(self, keyword: str = '백엔드', pages_to_crawl: int = 1,
is_newbie: bool = False):
        career_type = "1" if is_newbie else "" # 신입 필터 코드
        print(f"잡코리아에서 '신입' 필터: {is_newbie}, {pages_to_crawl} 페이지
까지 크롤링...")

        for page in range(1, pages_to_crawl + 1):
            search_url = f"{self.base_url}/Search/?stext={keyword}&Page_No=
{page}&careerType={career_type}"
            # ... (이하 목록 크롤링 로직은 동일) ...

    def get_job_description(self, url: str) -> Dict[str, str]:
        self.driver.get(url)
        self._random_sleep()
        soup = BeautifulSoup(self.driver.page_source, 'lxml')

        description = ""
        deadline = "상시채용" # 기본값

        try:
            # 본문 내용: 'detail-content' 클래스를 가진 div가 가장 안정적
            content_div = soup.select_one('div.detail-content')
            if content_div:
                description = content_div.text.strip()

            # 마감일: '오늘 마감' 또는 날짜 정보를 포함하는 span 찾기
            # 'd-day' 또는 'day' 클래스를 가진 span을 찾는다
            deadline_span = soup.find('span', class_=lambda c: c and ('d-
day' in c or 'day' in c))
            if deadline_span:
                deadline = deadline_span.text.strip()

        except Exception as e:
            print(f"  [잡코리아 상세 오류] {url} 처리 중: {e}")

        return {'description': description, 'deadline': deadline}
```

Step 3: `saramin_crawler.py` **최종 구현**

```
# crawlers/saramin_crawler.py (최종 버전)

# ...
    def crawl(self, keyword: str = '백엔드', pages_to_crawl: int = 1,
```

```
is_newbie: bool = False):
        exp_cd = "1" if is_newbie else "" # 신입 필터 코드
        print(f"사람인에서 '신입' 필터: {is_newbie}, {pages_to_crawl} 페이지까
지 크롤링...")

        for page in range(1, pages_to_crawl + 1):
            search_url = f"{self.base_url}/zf_user/search?searchword=
{keyword}&recruitPage={page}&exp_cd={exp_cd}"
            # ... (이하 목록 크롤링 로직은 동일) ...

    def get_job_description(self, url: str) -> Dict[str, str]:
        self.driver.get(url)
        self._random_sleep()
        soup = BeautifulSoup(self.driver.page_source, 'lxml')

        description_parts = []
        deadline = "상시채용"

        try:
            # 본문 내용: 'info-block' 클래스를 가진 모든 div의 텍스트를 합친다
            info_blocks = soup.select('div.info-block')
            for block in info_blocks:
                description_parts.append(block.text.strip())
            description = "\n\n".join(description_parts)

            # 마감일: 'end' 클래스를 가진 dt의 다음 형제(dd) 텍스트
            end_dt = soup.find('dt', class_='end')
            if end_dt and end_dt.find_next_sibling('dd'):
                deadline = end_dt.find_next_sibling('dd').text.strip()

        except Exception as e:
            print(f"  🕵️ [사람인 상세 오류] {url} 처리 중: {e}")

        return {'description': description, 'deadline': deadline}
```

Step 4: `main.py`와 Notion DB 최종 연동

1. **Notion DB:** `마감일` (텍스트 또는 날짜 유형) 속성을 새로 추가합니다.
2. `main.py` **수정:**

```
# main.py (최종 수정)

# ...
# 크롤러 실행 루프
crawlers_to_run = [WantedCrawler, JobKoreaCrawler, SaraminCrawler]
for CrawlerClass in crawlers_to_run:
    # ...
    # "신입" 공고만 가져오도록 is_newbie=True 설정!
    crawled_jobs = crawler.crawl(keyword='백엔드', pages_to_crawl=1,
is_newbie=True)
```

```
    for job in jobs_to_process: # 테스트 범위 제한된 리스트
        # ...
        # get_job_description은 이제 딕셔너리를 반환
        details = crawler.get_job_description(job_link)
        job['description'] = details.get('description', '')
        job['deadline'] = details.get('deadline', '상시채용')
# ...

# Notion 저장 루프
for job in all_jobs:
    # ... (중복 확인 및 필터링) ...

    deadline = job.get('deadline', '상시채용') # 마감일 정보 가져오기

    # ... (properties_to_save 딕셔너리 구성) ...
    if analysis_result:
        # ...

    properties_to_save['마감일'] = {'rich_text': [{'text': {'content':
deadline}}]}

    # notion.pages.create 호출
# ...
```

최종 행동 계획 (Next Action)

1. ☐ **`base_crawler.py`**의 `crawl`과 `get_job_description` 메서드 시그니처를 수정합니다.
2. ☐ **`jobkorea_crawler.py`**와 **`saramin_crawler.py`**에 위 최종 코드를 적용하여 모든 기능을 완성합니다. (`wanted_crawler.py`도 `get_job_description`이 딕셔너리를 반환하도록 수정해야 합니다.)
3. ☐ **Notion DB**에 `마감일` 속성을 추가합니다.
4. ☐ **`main.py`**를 최종 수정하여 '신입' 필터를 활성화하고, '마감일' 데이터를 수집하여 저장하도록 합니다.
5. ☐ **마지막 전체 테스트**를 실행하여, 3개 사이트의 '신입' 공고가 필터링되어 수집되고, 모든 정보(본문, 마감일, AI 분석)가 Notion에 완벽하게 저장되는지 확인합니다.