6장. 데이터베이스 이용

sqldf

sqldf 함수를 이용하면 데이터프레임의 데이터를 SQL 문장을 이용하여 조회할 수 있습니다.

```
sqldf(x, stringsAsFactors=FALSE)
```

구문에서...

- x : SELECT 구문을 작성합니다.
- stringAsFactors : TRUE인 경우 문자열을 팩터로 읽습니다. 기본값은 FALSE입니다.

SQL select 구문

다음 구문은 SQL SELECT 문장의 일반적인 구조입니다. SELECT 구문의 열 이름에 Sepal.Length처럼 점(.)이 포함돼 있으면 열 이름을 이중인용부호("Sepal.Length")로 묶어야 합니다.

```
SELECT [DISTINCT] { * ¦ 열이름 [[AS] 열별칭], ... }
FROM     데이터프레임이름
JOIN     데이터프레임이름
ON       조인조건
WHERE    조건
GROUP BY 그룹화열
HAVING   그룹조건
ORDER BY 정렬할열 [[ASC]¦DESC]
LIMIT [m,] n
```

중복행 제거

distinct는 중복된 행을 제거하여 한번만 출력되게 합니다.

```
> sqldf('select distinct Species from iris')
     Species
1     setosa
2 versicolor
3  virginica
```

데이터 조회 조건 지정

where 절은 데이터를 조회하는 조건을 지정할 수 있습니다.

```
> sqldf('select * from iris where Species="virginica"')
   Sepal.Length Sepal.Width Petal.Length Petal.Width   Species
1           6.3         3.3          6.0         2.5 virginica
2           5.8         2.7          5.1         1.9 virginica
3           7.1         3.0          5.9         2.1 virginica
4           6.3         2.9          5.6         1.8 virginica
...
```

그룹핑

group by를 이용하면 해당 팩터별로 집계할 수 있습니다. 다음 코드는 종별 Sepal.Length의 합(sum)을 출력합니다.

```
> sqldf('select Species, sum("Sepal.Length") as SepalLength from iris group by Species')
     Species SepalLength
1     setosa       250.3
2 versicolor       296.8
3  virginica       329.4
```

정렬

order by를 이용하면 정렬하여 데이터 조회할 수 있습니다. 기본 정렬은 오름차순(asc)입니다. desc는 내림차순으로 정렬합니다.

```
> sqldf('select Species, "Sepal.Length" from iris order by "Sepal.Length" limit 5')
  Species Sepal.Length
1  setosa          4.3
2  setosa          4.4
3  setosa          4.4
4  setosa          4.4
5  setosa          4.5
```

```
> sqldf('select Species, "Sepal.Length" from iris order by "Sepal.Length" desc limit 5')
    Species Sepal.Length
1 virginica          7.9
2 virginica          7.7
3 virginica          7.7
4 virginica          7.7
5 virginica          7.7
```

데이터 조회 개수 지정

limit는 일정 개수의 데이터만 출력해 줍니다. limit n 형식은 상위 n개 행을 출력하며, limit m, n 형식은 m번째 행부터 n개 행을 출력합니다. 이때 행 번호는 0부터 시작합니다.

```
> sqldf("select * from iris limit 5")
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
3          4.7         3.2          1.3         0.2  setosa
4          4.6         3.1          1.5         0.2  setosa
5          5.0         3.6          1.4         0.2  setosa
```

sqldf로 평균, 표준편차 구하기

다음 코드는 iris 데이터에서 종별 꽃 잎(Petal.Length)의 길이 평균과 표준편차를 출력합니다.

```
> sqldf('select avg("Petal.Length") as avg, stdev("Petal.Length") as sd from
iris group by Species')
    avg        sd
1 1.462 0.1736640
2 4.260 0.4699110
3 5.552 0.5518947
```

데이터 조인

두 개 이상 데이터셋을 이용하여 조인(JOIN)할 수 있습니다.[25]

```
> sqldf("select * from dataA join dataB on dataA.a1 == dataB.b1")
  a1 a2 a3 b1 b2
1  1  2  3  1  2
2  2  3  1  2  1
3  1  3  2  1  2
> dataA <- data.frame(a1=c(1, 2, 1), a2=c(2, 3, 3), a3=c(3, 1, 2))
> dataA
  a1 a2 a3
1  1  2  3
2  2  3  1
3  1  3  2
> dataB <- data.frame(b1=1:2, b2=2:1)
> dataB
  b1 b2
1  1  2
2  2  1
```

RJDBC

RJDBC는 R에서 JDBC를 이용하여 데이터베이스에 연결하기 위한 패키지입니다. RJDBC를 이용하여 데이터베이스에 연결하기 위해서 다음 일련의 작업들이 진행되어야 합니다.

1. 패키지 설치 및 로드
2. JDBC 드라이버 클래스 로드
3. 데이터베이스 연결
4. 테이블 데이터 조회
5. 테이블 데이터 수정
6. 데이터베이스 연결 종료

드라이버 클래스 로드

JDBC 드라이버 클래스파일이 있다면 데이터베이스에 연결하여 데이터를 조회하거나 조작할 수 있습니다. JDBC() 함수를 이용하면 JDBC 객체를 생성합니다. R의 JDBC 객체는 JDBCConnection 객체를 생성하기 위해 필요합니다.

```
JDBC(driverClass="", classPath="")
```

구문에서...

- driverClass="*driverClassFileName*" : JDBC 드라이더 클래스 파일 이름을 입력합니다. 오라클 데이터베이스에 접속하려면 'oracle.jdbc.OracleDriver"를 사용합니다.
- classPath="*driverClassPath*" : JDBC 드라이버 클래스 파일(jar 파일)의 위치[26]를 입력합니다. JDBC 드라이버클래스파일의 경로와 이름을 함께 입력해야 합니다. 만일 JDBC 드라이버 클래스파일이 현재 작업폴더(workplace)에 있다면 파일 이름만 입력해도 됩니다.

데이터베이스 연결

R의 JDBC 객체는 JDBCConnection 객체를 생성하기 위한 dbConnect() 함수를 가지고 있습니다. dbConnect() 함수는 JDBC 드라이버객체와 데이터베이스 접속 URL, 사용자 아이디와 비밀번호를 인자로 갖습니다.

```
dbConnect(drv, ...)
```

구문에서...

- *drv* : JDBC() 함수가 반환한 JDBC 드라이버 객체입니다.

테이블 데이터 조회

쿼리문을 실행시키기 위해서 dbGetQuery() 함수를 이용할 수 있습니다. dbGetQuery() 함수는 SELECT 쿼리용으로만 사용할 수 있습니다. 쿼리문의 실행 결과는 데이터 프레임으로 반환됩니다.

```
dbGetQuery(con, "statement", ...)
```

구문에서...

- *con* : dbConnect() 함수가 반환한 JDBCConnection 드라이버 객체입니다.
- *"statement"* : SELECT 쿼리문입니다.

테이블 전체 데이터 조회

테이블 전체 데이터를 가져오기 위해서라면 dbReadTable() 함수를 사용할 수 있습니다.

```
dbReadTable(con, "name", ...)
```

구문에서...

- *con* : dbConnect() 함수가 반환한 JDBCConnection 드라이버 객체입니다.
- "*name*" : 데이터베이스 테이블 이름을 입력합니다.

데이터 수정

데이터를 조작하기 위해서는 dbSendUpdate() 함수를 이용해야 합니다. dbSendUpdate() 함수는 어떤 결과 셋도 반환하지 않는 쿼리문을 실행시킬 때 사용합니다. dbExecute() 함수를 사용할 수 있지만 쿼리문에 이상이 없음에도 실행 시 에러가 출력될 때 dbSendUpdate() 함수는 좋은 대안으로 사용될 수 있습니다.

```
dbSendUpdate(con, "statement", ...)
```

사용 예

```
dbSendUpdate(con, "update employees set salary=30000 where employee_id=100")
```

데이터베이스 연결 종료

다음 구문은 데이터베이스 연결을 종료합니다.

```
dbDisconnect(con)
```