select 절

- select 절은 관계형 대수의 추출 연산에 대응한다. 질의의 결과로 바라는 애트리뷰트를 나열하는데 사용한다.
- loan 릴레이션내의 모든 지점명을 찾아라.

select *bn*
from *loan*

순수 관계형 대수 구문에서는 이 질의는 다음과 같다.

Π_{bn} (loan)

- select 절의 *는 "모든 애트리뷰트"를 의미한다.

select *
from *loan*

select 절(계속)

- SQL은 질의 결과와 함께 릴레이션내의 중복을 허용한다.
- 중복을 제거하려면 **select** 다음에 키워드 **distinct**를 기입한다.

- loan 릴레이션내의 모든 지점명을 찾아 중복은 제거하라.

- 키워드 all은 중복이 제거되지 않도록 한다.

select 절(계속)

- select 절에는 연산자 +,-,* 및 /를 내포한 산술 표현식과 상수 또는 튜플의 애트리뷰트 상의 연산을 내포할 수 있다.
- 질의:

  select *bn, ln, amount * 100*
  from *loan*

- 위의 질의는 애트리뷰트 amount에 100이 곱해진 것을 제외하고는 loan 릴레이션과 같은 릴레이션을 돌려준다.

where 절

- where 절은 관계형 대수의 선택 술어에 대응한다. from 절에 나타나는 릴레이션의 애트리뷰트를 내포하는 술어로 구성된다.

- 대출액이 1,200불을 초과하는 perryridge 지점에서 이루어진 대출의 대출 번호를 찾아라.

- SQL은 논리 연산자 and, or 및 not을 사용한다. SQL은 비교 연산자에 오퍼랜드로서 산술 표현식의 사용을 허용한다.

where 절(계속)

- SQL에는 어떤 값보다 작거나 같고 다른 값보다 크거나 같음을 나타내는 **where** 절을 단순히 하기 위해 **between** 비교 연산자를 포함한다.
- 대출액이 90,000불에서 100,000불 사이인 대출의 대출 번호를 찾아라.

```
select ln
from loan
where amount between 90000 and 100000
```

from 절

- **from** 절은 관계형 대수의 카티전 곱 연산에 대응한다. 표현식의 계산에서 검색될 릴레이션들을 나열한다.
- 카티전 곱 *borrower* × *loan* 을 찾아라.
- Perryridge 지점에 대출이 있는 모든 고객명과 대출번호를 찾아라.

재명명 연산

- 릴레이션과 애트리뷰트의 재명명을 위한 SQL 기법은 **as** 절로 이루어진다.

  old-name **as** *new-name*

- Perryridge 지점에 대출이 있는 모든 고객명과 대출 번호를 찾아라; 열 이름 ln을 loan-id로 대치하라.

튜플 변수

- 튜플 변수는 **as**절의 사용을 통해 **from**절에서 정의된다.

- Brooklyn에 위치한 어떤 지점보다 더 많은 자산을 가진 모든 지점명을 찾아라.

select distinct *T.bn*
from *branch* **as** *T, branch* **as** *S*
where *T.assets* > *S.assets* **and** *S.bc*= 'Brooklyn'

스트링 연산

- SQL에는 문자열 비교를 위한 문자열-매칭 연산자를 내포한다. 패턴은 두 개의 특수 문자를 사용해 기술한다.

  - %는 어떠한 부 문자열과 부합한다.

  - _는 어떤 문자 하나와 부합한다.

- 거리명에 문자열 “Main”을 내포한 모든 고객명을 찾아라.

튜플 출력의 순서화

- Perryridge 지점에 대출이 있는 모든 고객명을 알파벳 순서로 나열하라.

- 각 애트리뷰트에 대해 내림차순으로는 **desc**를 오름차순으로는 **asc**를 지정한다. 오름차순이 기본 값이다.
- SQL은 **order by** 요청을 받으면 정렬을 수행해야 한다. 많은 수의 튜플을 정렬하는데 비용이 많이 들어가므로, 필요할 때만 정렬하는 것이 바람직하다.

집합 연산

- 집합 연산 **union, intersect** 및 **except**는 릴레이션에 연산하며 관계형 대수 연산 ∪, ∩ 및 – 에 대응한다.
- 위의 각 연산은 자동으로 종복을 제거한다. 모든 중복을 유지하려면 상응하는 다중 집합 버전 **union all**, **intersect all** 및 **except all**을 사용한다. 어떤 튜플이 r에서 m번 나타나고 s에서 n번 나타난다고 가정하면 다음과 같이 나타난다.

  - r **union all** s 에 m + n 번
  - r **intersect all** s 에 min(m,n) 번
  - r **except all** s 에 max(0, m-n) 번

집합연산(계속)

- 대출, 예금 또는 모두를 가진 고객을 찾아라.

```
(select cn from depositor)
 union
(select cn from borrower)
```

- 대출과 예금을 모두 가진 고객을 찾아라.

```
SELECT distinct d.cn
from depositor d, borrower b
WHERE d.cn = b.cn
```

- 예금은 있으나 대출은 없는 고객을 찾아라.

```
SELECT distinct cn
from depositor
WHERE cn NOT IN (SELECT cn
                 FROM borrower)
```

집성 함수

- 이들 함수는 릴레이션의 행의 다중 집합 값에 연산하여 단일 값을 돌려준다.

avg: 평균 값
min: 최소 값
max: 최대 값
sum: 총 계
count: 값의 개수

집성함수(계속)

- Perryridge 지점의 평균 예금 잔고를 찾아라.
- customer 릴레이션의 튜플 수를 찾아라.
- 은행의 예금자 수를 찾아라.

집성함수 - Group By

- 각 지점별 예금자 수를 찾아라.

- 유의 : 집성 함수 외부의 **select** 절에 있는 애트리뷰트는 **group by** 리스트 내에 나타나야 한다.

집성함수 - Having 절

- 평균 예금 잔고가 1,200불을 초과하는 모든 지점명을 찾아라.

- 유의 : **having** 절의 술어는 그룹이 이루어진 후에 적용된다.

널 값(계속)

- loan 릴레이션 내의 amount에 널 값이 있는 모든 대출 번호를 찾아라.

  select *loan-number*
  from *loan*
  where *amount* **is null**

- 모든 대출액의 총계

  select sum *(amount)*
  from *loan*

- 위의 문장은 널 값은 무시한다. 널이 아닌 금액이 없으면 결과는 널이다.

- **count(*)**를 제외한 모든 집성 연산은 집성 애트리뷰트 상에 널 값을 가진 튜플은 무시한다.

중첩 부 질의

- SQL에서는 중첩 부 질의 기법을 제공한다.
- 부 질의는 다른 질의 내에 내포되는 select-from-where 표현식이다.
- 부 질의의 공통적인 사용은 집합 멤버쉽, 집합 비교 및 집합 수의 테스트를 수행하는 것이다.

집합 멤버쉽

- F **in** r $\Leftrightarrow \exists t \in r\ (t = F)$

(5 **in** | 0 | 4 | 5 |) = true

(5 **in** | 0 | 4 | 6 |) = false

(5 **not in** | 0 | 4 | 6 |) = true

예제 질의

- 은행에 예금과 대출이 모두 있는 고객을 찾아라.

- 은행에 대출은 있으나 예금은 없는 모든 고객을 찾아라.

예제 질의

- Perryridge 지점에 예금과 대출을 모두 가진 고객을 찾아라.

```
select distinct cn
from borrower, loan
where borrower.ln = loan.ln and
        bn = "Perryridge" and
        (bn, cn) in
            (select bn, cn
             from depositor, account
             where depositor.an =
                   account.an)
```

집합 비교

- Brooklyn에 위치한 어떤 지점보다 더 많은 자산을 가진 모든 지점을 찾아라.

select distinct *T.bn*
from *branch* **as** *T, branch* **as** *S*
where *T.assets > S.assets* **and** *S.bc=* 'Brooklyn'

some 절

- F<comp> some r $\Leftrightarrow$ $\exists$t(t $\in$ r $\wedge$ [F <comp> t])
- 여기서 <comp>는 다음 중 하나일 수 있다 : <>,=, <, ≤, >, ≥,

(5 < some { 0 | 5 | 6 }) = true

(다음과 같이 읽는다: 5 < 릴레이션내의 튜플)

(5 < some { 0 | 5 }) = false

(5 = some { 0 | 5 }) = true

(5 ≠ some { 0 | 5 }) = true (0 ≠ 5 이기때문에)

- (= **some**) ≡ **in**
- 그러나, (≠ **some**) ≡ **not in**

예제 질의

- Brooklyn에 위치한 어떤 지점보다 더 많은 자산을 가진 모든 지점을 찾아라.

all 절

- F<comp> **all** r ⇔ ∀t(t ∈ r ∧ [F <comp> t])

(5 < **all** | 0, 5, 6 |) = false

(5 < **all** | 6, 10 |) = true

(5 = **all** | 4, 5 |) = false

(5 ≠ all | 4, 6 |) = true (5 ≠ 4 이고 5 ≠ 6 이기 때문에)

- (≠ **all**) ≡ **not in**
- 그러나, (= **all**) ≡**in**

예제 질의

- Brooklyn에 위치한 모든 지점보다 더 많은 자산을 가진 모든 지점을 찾아라.

빈 릴레이션 검사

- **exists** 구조는 매개 변수 부 질의가 empty가 아니면 **참** 값을 돌려준다.
- **exists** r $\Leftrightarrow$ r $\neq \phi$
- **not exists** r $\Leftrightarrow$ r $= \phi$

예제 질의

- Brooklyn에 위치한 모든 지점에 예금이 있는 고객을 찾아라.

```
select distinct S.cn
from depositor as S
where not exists (
        (select bn
          from branch
          where bc = 'Brooklyn'
            AND bn NOT IN (select R.bn
                           from depositor T, account R
                           where T.an = R.an and S.cn = T.cn)))
```

- $X - Y = \phi \Leftrightarrow X \subseteq Y$ 임을 유의하라

중복 튜플의 부재 검사

- unique 구조는 부 질의가 그 결과 내에 중복 튜플을 가지고 있는지 여부를 검사한다.
- Perryridge 지점에 하나의 계좌만 가진 모든 고객을 찾아라.

```
select T.cn
from depositor as T
where unique (
        select R.cn
        from account, depositor as R
        where T.cn= R.cn
              and R.an = account.an
              and account.bn = "Perryridge")
```

중복 튜플의 부재 검사

- Perryridge 지점에 하나의 계좌만 가진 모든 고객을 찾아라.

```
select T.cn
from depositor T
WHERE 1 = (select count(R.cn)
           from account A, depositor R
           where T.cn = R.cn
                 and R.an = A.an
                 and A.bn = 'Perryridge')
```

예제 질의

- Perryridge 지점에 적어도 두 개의 계좌를 가진 모든 고객을 찾아라.

```
select distinct T.customer-name
from depositor T
where not unique (
        select R.customer-name
        from account, depositor as R
        where T.customer-name = R.customer-name and
                R.account-number = account.account-number and
                account.branch-name = "Perryridge")
```

예제 질의

- Perryridge 지점에 적어도 두 개의 계좌를 가진 모든 고객을 찾아라.

```
select T.cn
from depositor T
WHERE 2 >= (select count(R.cn)
            from account A, depositor R
            where T.cn = R.cn
                  and R.an = A.an
                  and A.bn = 'Perryridge')
```

유도 릴레이션

- 평균 예금 잔고가 600불을 초과하는 지점들의 평균 예금 잔고를 찾아라.

```
SELECT bn, mean
FROM (select bn, avg(balance) AS mean
                from account
                group by bn) AS result
WHERE mean > 600;
```

from 절 내에서 임시 릴레이션 result를 계산하고 애트리뷰트가 **where** 절에서 직접 사용될 수 있으므로, **having** 절을 사용할 필요가 없음에 유의하라.

뷰(view) 예제 질의

- 지점과 그들의 고객으로 구성된 뷰

```
create view allcustomer as
    (select bn, cn
     from depositor, account
     where depositor.an = account.an)
     union
    (select bn, cn
     from borrower, loan
     where borrower.ln = loan.ln)
```

- Perryridge 지점의 모든 고객을 찾아라.

```
select cn
from allcustomer
where bn = "Perryridge"
```

데이터베이스의 수정 - 갱신

- 10,000 불을 초과하는 모든 예금 계좌에는 6%를 다른 계좌에는 5%의 이자를 지급하라.

  - 두 개의 **update**문으로 작성하라.

update *account*
set *balance* = *balance* * 1.06
where *balance* > 10000

update *account*
set *balance* = *balance* * 1.05
where *balance* ≤ 10000

  - 순서가 중요하다.