Chapter 3 – Part 2 Review (Introduction to SQL (2))

Null Values and Three Valued Logic

true, false, unknown

Any comparison with *null* returns *unknown*

Example*: 5 < null or null <> null or null = null*

Three-valued logic using the *unknown*:

OR: (*unknown* **or** *true*) = *true*,
(*unknown* **or** *false*) = *unknown*
(*unknown* **or** *unknown)* = *unknown*

AND: *(true* **and** *unknown)* = *unknown,*
(false **and** *unknown)* = *false,*
(unknown **and** *unknown)* = *unknown*

NOT*: (***not** *unknown)* = *unknown*

"*P* **is unknown**" evaluates to true if predicate *P* evaluates to *unknown*

Result of **where** clause predicate is treated as *false* if it evaluates to *unknown*

Null Values and Three Valued Logic

Instructor

ID	Name	Dept_name	Salary
22222	Einstein	Physics	95000
32343	El Said	History	
33456	Gold	Physics	

SELECT name FROM instructor WHERE salary >100 and dept_name = ‘Physics”;

22222	**Einstein**	**Physics**	**95000**

SELECT name FROM instructor WHERE salary >100 or dept_name = ‘Physics”;

22222	**Einstein**	**Physics**	**95000**
33456	**Gold**	**Physics**	

Aggregate Functions (Cont.)

Find the average salary of instructors in the Computer Science department

select avg (*salary*)
from *instructor*
where *dept_name*= 'Comp. Sci.';

Find the total number of instructors who teach a course in the Spring 2010 semester

select count (**distinct** *ID*)
from *teaches*
where *semester* = 'Spring' **and** *year* = 2010

Find the number of tuples in the *course* relation

select count (*)
from *course*;

count(distinct *)는 못씀

Aggregate Functions – Group By

Find the average salary of instructors in each department

select *dept_name*, **avg** (*salary*)
from *instructor*
group by *dept_name*;

Note: departments with no instructor will not appear in result

ID	*name*	*dept_name*	*salary*
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

dept_name	*avg_salary*
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

Aggregate Functions – Group By

고객별로 주문한 도서의 총 수량과 총 판매액을 구하시오

```
SELECT custID, COUNT(*) AS 도서수량, SUM(price) AS 총액
 FROM orders
 GROUP BY custID;
```

orders

orderID	custID	bookID	price	order_date
2	1	3	21000	14/07/03
6	1	2	12000	14/07/07
1	1	1	6000	14/07/01
9	2	10	7000	14/07/09
3	2	5	8000	14/07/03
4	3	6	6000	14/07/04
10	3	8	13000	14/07/10
8	3	10	12000	14/07/08
7	4	8	13000	14/07/07
5	4	7	20000	14/07/05

custID	도서수량	총액
1	3	39000
2	2	15000
3	3	31000
4	2	33000

Aggregate Functions – Having Clause

Find the names and average salaries of all departments whose average salary is greater than 42000

select *dept_name*, **avg** (*salary*)
from *instructor*
group by *dept_name*
having avg (*salary*) > 42000;

Note: predicates in the **having** clause are applied after the formation of groups whereas predicates in the **where** clause are applied before forming groups

```
SQL> select dept_name, count(*) from instructor where salary >1000 group by dept_name having count(*)>1;

DEPT_NAME                            COUNT(*)
------------------------------------ ----------
Physics                                       2
Comp. Sci.                                    3
Finance                                       2
History                                       2
```

3.8 Nested Subqueries

SQL provides a mechanism for the nesting of subqueries.

A **subquery** is a **select-from-where** expression that is nested within another query.

A common use of subqueries is to perform tests for set membership, set comparisons, and set cardinality.

Nested Subqueries - Set Membership

Customer

custID	Name
1	박지성
2	김연아
3	장미란
4	추신수
5	박세리

Orders

orderID	custID
2	1
6	2
1	1
9	3
3	2
4	3
10	4

select *name*
from *customer*
where *custID* **in** (**select** *custID* **from** *orders*);

IN에서 사용가능 한 subquery는
결과로 다중 행, 다중열을 반환할 수 있음

* Equivalent
→ **select** *name* **from** *customer*
where *custID = 1 or custID = 2 or custID = 3 or custID = 4*;

Nested Subqueries - Set Membership

Find courses offered in Fall 2009 and in Spring 2010

```
select distinct course_id
from section
where semester = 'Fall' and year= 2009 and
        course_id   in (select course_id
                        from section
                        where semester = 'Spring' and year= 2010);
```

Find courses offered in Fall 2009 but not in Spring 2010

```
select distinct course_id
from section
where semester = 'Fall' and year= 2009 and
        course_id   not in (select course_id
                            from section
                            where semester = 'Spring' and year= 2010);
```

Nested Subqueries - Set Comparison

Find names of instructors with salary greater than that of some (at least one) instructor in the Biology department.

```
select distinct T.name
from instructor as T, instructor as S
where T.salary  > S.salary and S.dept_name = 'Biology';
```

name	salary	Department
Lee	5000	Biology
Kim	10000	Biology
Park	7000	Computer

Same query using > **some** clause

```
select name
from instructor
where salary > some (select salary
                     from instructor
                     where dept_name = 'Biology');
```

→ (5000, 10000)

결과는 ? Kim, Park

Example Query of all Clause

Find the names of all instructors whose salary is greater than the salary of all instructors in the Biology department.

name	salary	Department
Lee	5000	Biology
Kim	10000	Biology
Park	7000	Computer

```
select name
from instructor
where salary > all (select salary
                    from instructor
                    where dept_name = 'Biology');
```

결과는 ? No rows selected!!

Exists

Customer

custID	Name
1	박지성
2	김연아
3	장미란
4	추신수
5	박세리

Orders

orderID	custID
2	1
6	2
1	1
9	3
3	2
4	3
10	4

```
select name
from customer cs
where exists (select custID
              from orders od
              where cs.custID = od.custID);
```

custID	Name
1	박지성
2	김연아
3	장미란
4	추신수

Exists

IN과 EXISTS의 차이

EXISTS : 단지 해당 row가 존재하는지만 확인하고, 더 이상 수행되지 않음 (True or False로 값을 Return)

IN : 실제 존재하는 데이터들의 모든 값까지 확인함

(다중행 및 다중열로 값을 Return)

→ 일반적인 경우에 EXISTS가 더 좋은 성능 나타냄

EXISTS 실행 순서

1. **Customer cs** 테이블에서 첫 행을 가지고 온 후 **Subquery**에 **cs** 값으로 입력
2. **Subquery**에서 **Order od** 테이블의 어떤 행에서 **cs**의 **custID**와 같은것을 찾으면 **EXISTS**는 **True**를 **Return**
3. 이후 **cs**테이블에서 첫 행에 대한 **name**이 반환

Customer cs 테이블에서 나머지 행에 대해서 반복

Exists

Yet another way of specifying the query “Find all courses taught in both the Fall 2009 semester and in the Spring 2010 semester”

바깥질의

```
select course_id
from section as S
where semester = 'Fall' and year= 2009 and
      exists (select *
              from section as T
              where semester = 'Spring' and year= 2010
                    and S.course_id= T.course_id);
```

하위질의

correlated

Correlated subquery

Correlation name or **correlation variable**

Subqueries in the From Clause

SQL allows a subquery expression to be used in the **from** clause

Find the average instructors' salaries of those departments where the average salary is greater than $42,000.

```
select dept_name, avg_salary
from ( select dept_name, avg (salary) as avg_salary
       from instructor
       group by dept_name)
where avg_salary > 42000;
```

select-from-where 결과가 결국 또 다른 relation이기 때문에 가능한 것임

Note that we do not need to use the **having** clause → 바깥쪽에 where 절 때문

Another way to write above query

```
select dept_name, avg_salary
from ( select dept_name, avg (salary)
       from instructor
       group by dept_name)
       as dept_avg (dept_name, avg_salary)
where avg_salary > 42000;
```

from 절 안에 있는 select 구문의 결과를 새로운 relation (즉, dept_avg)로 재명명
→ Oracle에서는 지원되지 않음

With Clause

The **with** clause provides a way of defining a temporary view whose definition is available only to the query in which the **with** clause occurs.

Find all departments with the maximum budget

```
with max_budget (value) as
    (select max(budget)
     from department)
select budget
from department, max_budget
where department.budget = max_budget.value;
```

클라이언트와의 연결이 끊어지면 데이터가 사라짐

with절을 포함한 질의에서만 유효한 임시 relation (*max_budget*) 생성

기존을 with절을 사용해 질의를 변형시키는 문제나옴

select * from tab; //유저가 생성한 모든 테이블을 보는 질의

Scalar Subquery

Scalar subquery is one which is used where a single value is expected

스칼라 값은 벡터 값에 대응되는 말로 단일값을 의미함!!

E.g.

```
select dept_name,
    (select count(*)
     from instructor
     where department.dept_name = instructor.dept_name)
     as num_instructors
from department;
```

Subqeury의 결과가 table이 아니라 value인 경우

질의의 의미 ? → 학과별 교수님의 수를 출력

Deletion (Cont.)

Delete all instructors whose salary is less than the average salary of instructors

delete from *instructor*
where *salary*< (**select avg** (*salary*) **from** *instructor*);

Problem: as we delete tuples from instructor, the average salary changes

Solution used in SQL:

1. First, compute **avg** salary and find all tuples to delete
2. Next, delete all tuples found above (without recomputing **avg** or retesting the tuples)

Insertion (Cont.)

Add all instructors to the *student* relation with tot_creds set to 0

insert into *student*

select *ID, name, dept_name, 0*
from *instructor*

insert 수행전에 select가 먼저 수행되어야 함

The **select from where** statement is evaluated fully before any of its results are inserted into the relation (otherwise queries like

insert into *table*1 **select** * **from** *table*1

would cause problems, if *table1* did not have any primary key defined)

만약 select from where 구문을 수행하면서 insert문을 수행하면 어떻게 될까?
즉, select를 해서 하나의 tuple을 뽑고 이를 다시 같은 table에 insert를 하고,
주키 제약조건마저 없다면 어떻게 될까 ?

→ 중복된 tuple들이 계속해서 들어가고 결국 무한 loop에 빠진다.

Modification of the Database – Updates

Increase salaries of instructors whose salary is over $100,000 by 3%, and all others receive a 5% raise

Write two **update** statements:

①
```
update instructor
set salary = salary * 1.03
where salary > 100000;
```

②
```
update instructor
set salary = salary * 1.05
where salary <= 100000;
```

②→① 순서로 수행되면 ?

$100000 보단 약간 적은 연봉 (즉, 5% 인상하면 $100000가 넘는 연봉)을 받는 사람들은 다시 3% 더 인상됨 → 본래 의도?

The order is important

Can be done better using the **case** statement (next slide)

Case Statement for Conditional Updates

Same query as before but with case statement

```
update instructor
set salary = case
                when salary <= 100000 then salary * 1.05
                else salary * 1.03
             end
```