CHAPTER 4

Intermediate SQL

In this chapter, we continue our study of SQL. We consider more complex forms of SQL queries, view definition, transactions, integrity constraints, more details regarding SQL data definition, and authorization.

As in Chapter 3 exercises, students should be encouraged to execute their queries on the sample database provided on http://db-book.com, to check if they generate the expected answers. Students could even be encouraged to create sample databases that can expose errors in queries, for example where an inner join operation is used erroneously in place of an outerjoin operation.

Exercises

4.12 For the database of Figure 4.11, write a query to find those employees with no manager. Note that an employee may simply have no manager listed or may have a *null* manager. Write your query using an outer join and then write it again using no outer join at all.

Answer:

a.

select *employee_name*
from *employee* **natural left outer join** *manages*
where *manager_name* **is null**

employee (*employee_name*, *street*, *city*)
works (*employee_name*, *company_name*, *salary*)
company (*company_name*, *city*)
manages (*employee_name*, *manager_name*)

Figure 4.11 Employee database for Figure 4.7 and 4.12.

b.

```
select employee_name
from employee e
where not exists
    (select employee_name
    from manages m
    where e.employee_name = m.employee_name and
            m.manager_name is not null)
```

4.13 Under what circumstances would the query

```
select *
from student natural full outer join takes
                    natural full outer join course
```

include tuples with null values for the *title* attribute?

Answer: We first rewrite the expression with parentheses to make clear the order of the left outer join operations (the SQL standard specifies that the join operations are left associative).

```
select *
from (student natural full outer join
    takes) natural full outer join course
```

Given the above query, there are 2 cases for which the *title* attribute is null

a. Since *course_id* is a foreign key in the *takes* table referencing the *course* table, the title attribute in any tuple obtained from the above query can be null if there is a course in *course* table that has a null title.

b. If a student has not taken any course, as it is a **natural full outer join**, such a student's entry would appear in the result with a **null** *title* entry.

4.14 Show how to define a view *tot_credits* (*year, num_credits*), giving the total number of credits taken by students in each year.

Answer:

```
create view tot_credits(year, tot_credits)
as
    (select year, sum(credits)
    from takes natural join course
    group by year)
```

Note that this solution assumes that there is no year where students didn't take any course, even though sections were offerred.

salaried_worker (*name*, *office*, *phone*, *salary*)
hourly_worker (*name*, *hourly_wage*)
address (*name*, *street*, *city*)

Figure 4.12 Employee database for Exercise 4.16.

4.15 Show how to express the **coalesce** operation from Exercise 4.10 using the **case** operation.

Answer:

select
  case *Result*
    when (A_1 is not null) **then** A_1
    when (A_2 in not null) **then** A_2
    .
    .
    .
    when (A_n is not nulll) **then** A_n
    else null
  end
from *A*

4.16 Referential-integrity constraints as defined in this chapter involve exactly two relations. Consider a database that includes the relations shown in Figure 4.12. Suppose that we wish to require that every name that appears in *address* appears in either *salaried_worker* or *hourly_worker*, but not necessarily in both.

a. Propose a syntax for expressing such constraints.

b. Discuss the actions that the system must take to enforce a constraint of this form.

Answer:

a. For simplicity, we present a variant of the SQL syntax. As part of the **create table** expression for *address* we include

foreign key (*name*) **references** *salaried_worker* **or** *hourly_worker*

b. To enforce this constraint, whenever a tuple is inserted into the *address* relation, a lookup on the *name* value must be made on the *salaried_worker* relation and (if that lookup failed) on the *hourly_worker* relation (or vice-versa).

4.17 Explain why, when a manager, say Satoshi, grants an authorization, the grant should be done by the manager role, rather than by the user Satoshi.

Answer: Consider the case where the authorization is provided by the user Satoshi and not the manager role. If we revoke the authorization from Satoshi, for example because Satoshi left the company, all authorizations that Satoshi had granted would also be revoked, even if the grant was to an employee whose job has not changed.
If the grant is done by the manager role, revoking authorizations from Satoshi will not result in such cascading revocation.
In terms of the authorization graph, we can treat Satoshi and the role manager as nodes. When the grant is from the manager role, revoking the manager role from Satoshi has no effect on the grants from the manager role.

4.18 Suppose user A, who has all authorizations on a relation r, grants select on relation r to **public** with grant option. Suppose user B then grants select on r to A. Does this cause a cycle in the authorization graph? Explain why.
Answer: Yes, it does cause a cycle in the authorization graph. The grant to public results in an edge from A to public. The grant to the **public** operator provides authorization to everyone, B is now authorized. For each privilege granted to *public*, an edge must therefore be placed between *public* and all users in the system. If this is not done, then the user will not have a path from the root (DBA). And given the with grant option, B can grant select on r to A resultin in an edge from B to A in the authorization graph. Thus, there is now a cycle from A to public, from public to B, and from B back to A.

4.19 Database systems that store each relation in a separate operating-system file may use the operating system's authorization scheme, instead of defining a special scheme themselves. Discuss an advantage and a disadvantage of such an approach.
Answer:

- Advantages:
  - Operations on the database are speeded up as the authorization procedure is caried out at the OS level.
  - No need to implement the security and authorization inside the DBMS. This may result in reduced cost.
  - Administrators need not learn new commands or use of a new UI. They can create and administer user accounts on the OS they may already be familiar with.
  - No worry of unauthorized database users having direct access to the OS files and thus bypassing the database security.
- Disadvantages:
  - Database users must correspond to operating system users.
  - Fine control on authorizations is limited by what the operating system provides and is dependent on it, For example, most operating systems

do not distinguish between insert, update and delete, they just have a coarse level privilege called "modify". Privileges such as "references" cannot be provided.

- Columnwise control is not possible. You cannot differentiate update/delete and insert authorizations.
- Cannot store more than one relation in a file.
- The with grant option is limited to what the OS provides (if any: most OS's don't provide such options) and cannot be controlled by the user or administrator.