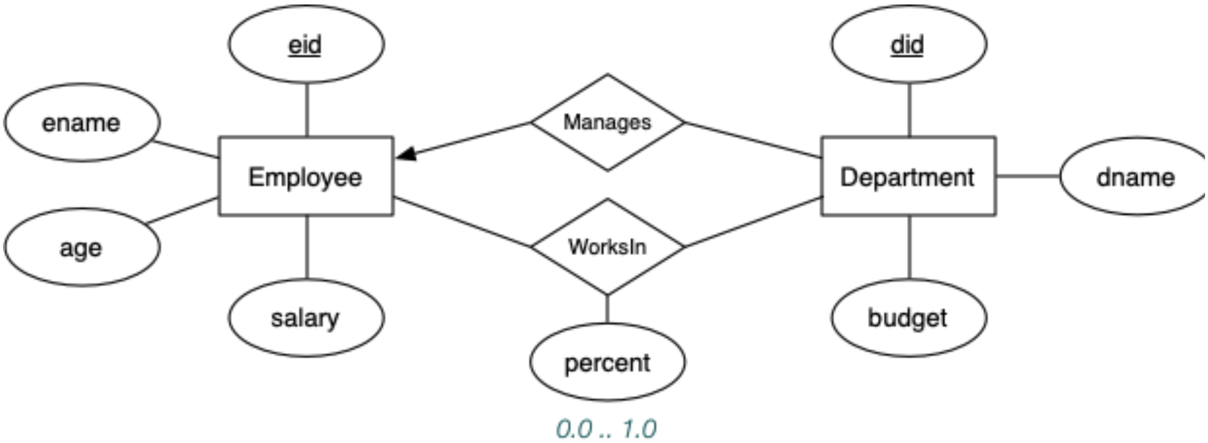
COMP3311 22T1 Database Systems

Week 04
SQL Constraints, Updates and Queries

Consider the following data model for a a business organisation and its employees:



Employees are uniquely indentified by an id (`eid`), and other obvious information (name,age,...) is recorded about each employee. An employee may work in several departments, with the percentage of time spent in each department being recorded in the `WorksIn` relation (as a number in the range 0.0-1.0, with 1.0 representing 100%). The percentages for a given employee may not sum to one if the employee only works part-time in the organisation. Departments are also uniquely identified by an id (`did`), along with other relevant information, including the id of the employee who manages the department.

Based on the ER design and the above considerations, here is a relational schema to represent this scenario:

```
create table Employees (
      eid     integer,
      ename   text,
      age     integer,
      salary  real,
      primary key (eid)
);
create table Departments (
      did     integer,
      dname   text,
      budget  real,
      manager integer references Employees(eid),
      primary key (did)
);
create table WorksIn (
      eid     integer references Employees(eid),
      did     integer references Departments(did),
      percent real,
      primary key (eid,did)
```

```
);
```

Answer each of the following questions for this schema ...

1. Does the order of table declarations above matter?

   Answer:

   Yes. The order matters. We can not insert a Department tuple until there is an Employee tuple available to be the manager of the department. We cannot also insert any WorksIn tuple until you have both the Employee tuple and the Department tuple where the employee works.

2. A new government initiative to get more young people into work cuts the salary levels of all workers under 25 by 20%. Write an SQL statement to implement this policy change.

   Answer:

   SQL statement to reduce pay for people under 25 by 20%:

   ```
   update Employees
   set    salary = salary * 0.8
   where  age < 25;
   ```

   A straightforward applicatiom of the SQL `UPDATE` statement.

3. The company has several years of growth and high profits, and considers that the Sales department is primarily responsible for this. Write an SQL statement to give all employees in the Sales department a 10% pay rise.

   Answer:

   SQL statement to give Sales employees a 10% pay rise:

   ```
   update Employees e
   set    e.salary = e.salary * 1.10
   where  eid in
              (select eid
               from   Departments d, WorksIn w
               where  d.dname = 'Sales' and d.did = w.did
              );
   ```

   This query requires that we know which department an employee works for before updating their pay. The only information we have in the `Employees` relation to help with this is the employee id. Thus the subquery gives a list of ids for all employees working in the Sales department.

4. Add a constraint to the `CREATE TABLE` statements above to ensure that every department must have a manager.

   Answer:

```
   create table Departments (
      did       integer,
      dname     varchar(20),
      budget    real,
      manager   integer not null,
      primary key (did),
      foreign key (manager) references Employees(eid)
   );
```

Change the definition of `Departments` to ensure that the foreign key `manager` is not null. The foreign key constraint already ensures that the value must be a primary key in the `Employees` relation but, without the `NOT NULL` declaration, a foreign key value is allowed to be null.

5. Add a constraint to the `CREATE TABLE` statements above to ensure that no-one is paid less than the minimum wage of $15,000.

   Answer:

   This is a straightforward example of a single-attribute constraint. It is effectively a constraint on the domain of the `salary` attribute. In fact, it would been sensible to have an initial constraint to ensure that the `salary` was at least positive. A similar constraint should probably be applied to the `percent` attribute in the `WorksIn` relation.

```
create table Employees (
   eid       integer,
   ename     varchar(30),
   age       integer,
   salary    real check (salary >= 15000),
   primary key (eid)
);
```

6. Add a constraint to the `CREATE TABLE` statements above to ensure that no employee can be committed for more than 100% of his/her time. Note that the SQL standard allows queries to be used in constraints, even though DBMSs don't implement this (for performance reasons).

   Answer:

   We have expressed this as a tuple-level constraint, even though it's really a constraint on the `eid` attribute. Note the use of scoping in the sub-query: the `eid` refers to the id of the employee that is being inserted or modified, while `w.eid` is bound by the tuple variable in the subquery. The subquery itself computes the total `percent` that the employee `eid` works.

   This query is valid according to Ullman/Widom's description of the SQL2 standard. However, neither PostgreSQL nor Oracle supports subqueries in `CHECK` conditions. The condition can only be an expression involving the attributes in the updated row.

```
create table WorksIn (
   eid         integer,
   did         integer,
   percent    real,
   primary key (eid,did),
   foreign key (eid) references Employees(eid),
   foreign key (did) references Departments(did)
   constraint  MaxFullTimeCheck
               check (1.00 >= (select sum(w.percent)
                               from   WorksIn w
                               where  w.eid = eid)
                      )
);
```

In most relational database management systems, the constraint checking required here would need to be implemented via a trigger.

7. Add a constraint to the `CREATE TABLE` statements above to ensure that a manager works 100% of the time in the department that he/she manages. Note that the SQL standard allows queries to be used in constraints, even though DBMSs don't implement this (for performance reasons).

   Answer:

   We have expressed this as a tuple-level constraint, even though it's really a constraint on the `manager` attribute.

```
create table Departments (
   did         integer,
   dname       varchar(20),
   budget      real,
   manager     integer,
   primary key (did),
   foreign key (manager) references Employees(eid)
   constraint  FullTimeManager
               check (1.0 = (select w.percent
                             from   WorksIn w
                             where  w.eid = manager)
                     )
);
```

   As in the previous question, this kind of constraint is allowed by the SQL standard, but DBMSs typically don't implement cross-table constraints like this; a trigger is required and the check has to be programmed in the trigger.

8. When an employee is removed from the database, it makes sense to also delete all of the records that show which departments he/she works for. Modify the `CREATE TABLE` statements above to ensure that this occurs.

   Answer:

The `ON DELETE CASCADE` clause ensures that when the `Employees` record for `eid` is removed, then any `WorksIn` tuples that refer to `eid` are also removed.

```
create table WorksIn (
   eid         integer,
   did         integer,
   percent    real,
   primary key (eid,did),
   foreign key (eid) references Employees(eid) on delete
   foreign key (did) references Departments(did)
);
```

Of course, this immediately reaises the issue of references to the `Departments` relation; this is considered in the next question.

9. When a manager leaves the company, there may be a period before a new manager is appointed for a department. Modify the `CREATE TABLE` statements above to allow for this.

   Answer:

   If the department has no manager, we indicate this by putting a value of `NULL` for the `manager` field. However, in one of the questions above, we added a NOT NULL contraint to ensure that every department *does* have a manager. To solve this question, we need to remove that constraint.

   An alternative would be to always appoint a temporary manager, which could be accomplished via an `UPDATE` statement, e.g.

   ```
   update department set manager = SomeEID where did = OurDe
   ```

10. Consider the deletion of a department from a database based on this schema. What are the options for dealing with referential integrity between `Departments` and `WorksIn`? For each option, describe the required behaviour in SQL.

    Answer:

    Three possible approaches to referential integrity between between `Departments` and `WorksIn`:

    a. Disallow the deletion of a `Departments` tuple if any `Works` tuple refers to it. This is the default behaviour, which would result from the `CREATE TABLE` definition in the previous question.

    b. When a `Departments` tuple is deleted, also delete all `WorksIn` tuples that refer to it. This requires adding an `ON DELETE CASCADE` clause to the definition of `WorksIn`.

    ```
    create table WorksIn (
       eid     integer,
    ```

```
   did     integer,
   percent real,
   primary key (eid,did),
   foreign key (eid) references Employees(eid) on del
   foreign key (did) references Departments(did) on (
);
```

In this solution, we've added the same functionality to the `eid` field as well (see previous question).

c. For every `WorksIn` tuple that refers to the deleted department, set the `did` field to the department id of some existing 'default' department. Unfortunately, Oracle doesn't appear to implement this functionality. If it did, the definition of `WorksIn` would change to:

```
create table WorksIn (
   eid     integer,
   did     integer default 1,
   percent real,
   primary key (eid,did),
   foreign key (eid) references Employees(eid) on del
   foreign key (did) references Departments(did) on (
);
```

11. For each of the possible cases in the previous question, show how deletion of the Engineering department would affect the following database:

```
  EID ENAME                AGE     SALARY
----- --------------- ----- ----------
    1 John Smith          26      25000
    2 Jane Doe            40      55000
    3 Jack Jones          55      35000
    4 Superman            35      90000
    5 Jim James           20      20000

  DID DNAME                 BUDGET  MANAGER
----- --------------- ---------- --------
    1 Sales                500000        2
    2 Engineering         1000000        4
    3 Service              200000        4

  EID   DID  PCT_TIME
----- ----- ---------
    1     2      1.00
    2     1      1.00
    3     1      0.50
    3     3      0.50
    4     2      0.50
    4     3      0.50
    5     2      0.75
```

Answer:

a. Disallow ... The database would not change. The DBMS would print an error message about referential integrity constraint violation.

b. `ON DELETE CASCADE` ... All of the tuples in the `WorksIn` relation that have `did = 2` are removed, giving:

```
  DID DNAME                 BUDGET  MANAGER
----- --------------- ---------- --------
    1 Sales                 500000        2
    3 Service               200000        4

  EID   DID  PCT_TIME
----- ----- ---------
    2     1      1.00
    3     1      0.50
    3     3      0.50
    4     3      0.50
```

c. `ON DELETE SET NULL` ... All of the tuples in the `WorksIn` relation that have `did = 2` have that attribute modified to `NULL`, giving:

```
  DID DNAME                 BUDGET  MANAGER
----- --------------- ---------- --------
    1 Sales                 500000        2
    3 Service               200000        4

  EID   DID  PCT_TIME
----- ----- ---------
    1  NULL      1.00
    2     1      1.00
    3     1      0.50
    3     3      0.50
    4  NULL      0.50
    4     3      0.50
    5  NULL      0.75
```

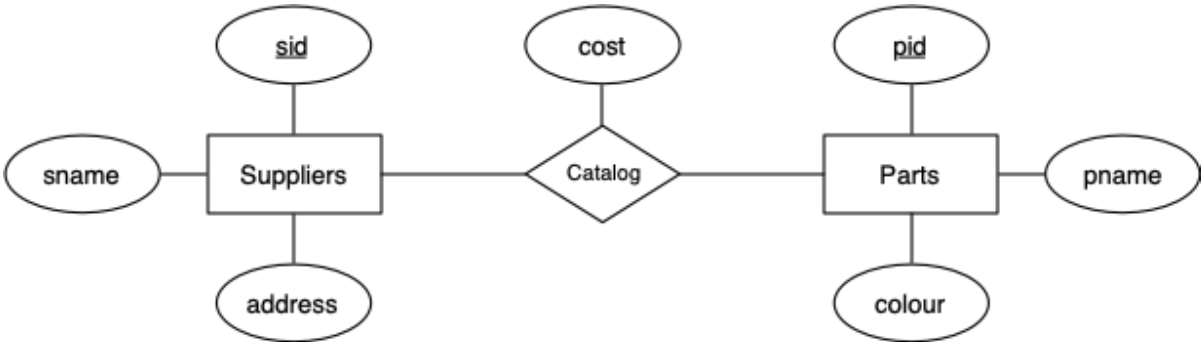
d. `ON DELETE SET DEFAULT` ... All of the tuples in the `WorksIn` relation that have `did = 2` have that attribute modified to the default department (`1`), giving:

```
  DID DNAME                 BUDGET  MANAGER
----- --------------- ---------- --------
    1 Sales                 500000        2
    3 Service               200000        4

  EID   DID  PCT_TIME
----- ----- ---------
    1     1      1.00
    2     1      1.00
    3     1      0.50
    3     3      0.50
    4     1      0.50
```

```
    4      3      0.50
    5      1      0.75
```

Consider the following data model for a a business that supplies various parts:



Based on the ER design and the above considerations, here is a relational schema to represent this scenario:

```
create table Suppliers (
      sid     integer primary key,
      sname   text,
      address text
);
create table Parts (
      pid     integer primary key,
      pname   text,
      colour  text
);
create table Catalog (
      sid     integer references Suppliers(sid),
      pid     integer references Parts(pid),
      cost    real,
      primary key (sid,pid)
);
```

Write SQL statements to answer each of the following queries ...

> **Note1:** all of these solutions have alternative formulations. If you think you have a better solution than the one(s) presented here, let me know.
> **Note2:** a useful strategy, when developing an SQL solution to an information request, is to express intermediate results as views; this has been done in a few solutions here, but you might like to consider reformulating more of them with views, for clarity.

12. Find the *names* of suppliers who supply some red part.

Answer:

```
select S.sname
from   Suppliers S, Parts P, Catalog C
where  P.colour='red' and C.pid=P.pid and C.sid=S.sid
```

or

```
select sname
from   Suppliers natural join Catalog natural join Parts
where  P.colour='red'
```

13. Find the *sids* of suppliers who supply some red or green part.

    Answer:

```
select C.sid
from   Parts P, Catalog C
where  (P.colour='red' or P.colour='green') and C.pid=P.p
```

or

```
select sid
from   Catalog C natural join Parts P
where  (P.colour='red' or P.colour='green')
```

14. Find the *sids* of suppliers who supply some red part or whose address is 221 Packer Street.

    Answer:

```
select S.sid
from   Suppliers S
where  S.address='221 Packer Street'
       or S.sid in (select C.sid
                    from   Parts P, Catalog C
                    where  P.color='red' and P.pid=C.pid
                   )
```

15. Find the *sids* of suppliers who supply some red part and some green part.

    Answer:

```
select C.sid
from   Parts P, Catalog C
where  P.color='red' and P.pid=C.pid
       and exists (select P2.pid
                   from   Parts P2, Catalog C2
                   where  P2.color='green' and C2.sid=C.s
                  )
```

or

```
(select C.sid
 from Parts P, Catalog C
 where P.color='red' and P.pid=C.pid
)
intersect
(select C.sid
 from Parts P, Catalog C
 where P.color='green' and P.pid=C.pid
)
```

16. Find the *sids* of suppliers who supply every part.

Answer:

```
select S.sid
from   Suppliers S
where  not exists((select P.pid from Parts P)
                  except
                  (select C.pid from Catalog C where C.si
                 )
```

or

```
select C.sid
from   Catalog C
where  not exists(select P.pid
                  from   Part P
                  where  not exists(select C1.sid
                                    from   Catalog C1
                                    where  C1.sid=C.sid a
                                   )
                 )
```

17. Find the *sids* of suppliers who supply every red part.

Answer:

```
select S.sid
from   Suppliers S
where  not exists((select P.pid from Parts P where P.colo
                  except
                  (select C.pid from Catalog C where C.si
                 )
```

or

```
select C.sid
from   Catalog C
where  not exists(select P.pid
                  from   Part P
                  where  P.colour='red' and
                         not exists(select C1.sid
                                    from   Catalog C1
                                    where  C1.sid=C.sid a
                                   )
                 )
```

18. Find the *sids* of suppliers who supply every red or green part.

Answer:

```
select S.sid
from   Suppliers S
where  not exists((select P.pid from Parts P
                   where P.color='red' or P.color='green
                  except
                  (select C.pid from Catalog C where C.si
                 )
```

or

```
select C.sid
from   Catalog C
where  not exists(select P.pid
                  from   Part P
                  where  (P.colour='red' or P.colour='gre
                         not exists(select C1.sid
                                    from   Catalog C1
                                    where  C1.sid=C.sid a
                                   )
                 )
```

19. Find the *sids* of suppliers who supply every red part or supply every green part.

Answer:

```
(select S.sid
 from   Suppliers S
 where  not exists((select P.pid from Parts P where P.col
                   except
                   (select C.pid from Catalog C where C.s
                  )
)
union
(select S.sid
 from   Suppliers S
 where  not exists((select P.pid from Parts P where P.col
                   except
                   (select C.pid from Catalog C where C.s
                  )
)
```

20. Find pairs of *sids* such that the supplier with the first *sid* charges more for some part than the supplier with the second *sid*.

Answer:

```
select C1.sid, C2.sid
from   Catalog C1, Catalog C2
where  C1.pid = C2.pid and C1.sid != C2.sid and C1.cost >
```

21. Find the *pids* of parts that are supplied by at least two different suppliers.

Answer:

```
select C.pid
from   Catalog C
where  exists(select C1.sid
              from   Catalog C1
              where  C1.pid = C.pid and C1.sid != C.sid
             )
```

22. Find the *pids* of the most expensive part(s) supplied by suppliers named

"Yosemite Sham".

Answer:

```
select C.pid
from   Catalog C, Suppliers S
where  S.sname='Yosemite Sham' and C.sid=S.sid and
       C.cost >= all(select C2.cost
                     from   Catalog C2, Suppliers S2
                     where  S2.sname='Yosemite Sham' and
                    )
```

23. Find the *pids* of parts supplied by every supplier at a price less than 200 dollars (if any supplier either does not supply the part or charges more than 200 dollars for it, the part should not be selected).

Answer:

```
select C.pid
from   Catalog C
where  C.price < 200.00
group by C.pid
having count(*) = (select count(*) from Suppliers);
```