Process to Prepare Deleting Duplicate Employees

STEP 1:

As is customary, identify the database that contains the data followed by the code to allow for grouping data as needed.

```
use employees;
SET SESSION sql_mode=(SELECT REPLACE(@@sql_mode,'ONLY_FULL_GROUP_BY,',''));
```

Identify the duplicate entries in the database. The criteria for a duplicate entry is one with two or more employees with the same exact full name, address, and company email address. The query that captures the duplicate entries is:

```
select employee.eid, efn, eln, estreet, ecity, est, ezip,
   contact_id, contact, pos_id, salary, comments, status_id, start_date
from Employee join EmpHistory on Employee.eid = EmpHistory.eid
          join Address on Address.eid = Employee.eid
                    join EmpContact on EmpContact.eid = Employee.eid
where contact_id = 2
group by contact
having COUNT(*)>1
order by contact;
```

There are 33 rows return with this query indicating there are grouped two records for each of the 33 employees returned in the query. To be sure the correct employee record gets deleted, the next query returns 66 records, two records for each of the employees identified as having a duplicate entry in the database. To break out these records into the 66 we need to create a temporary table to hold these 33 records.

STEP 2:

The next step is to query the database extracting out employee data and set the **where** clause such that the email address in the EmpContact table is exactly the same as the email address held in the temporary table. Name the temporary table **emps**.

To begin , create the temporary table using the code below:

```
drop temporary table if exists emps;
create temporary table if not exists emps(
eid int NULL,
efn varchar(45) NULL,
eln varchar(45) NULL,
estreet varchar(45) NULL,
ecity varchar(45) NULL,
est char(2) NULL,
ezip char(5) NULL,
contact_id int NULL,
email varchar(255) NULL,
phone varchar(255) NULL,
pos_id int NULL,
salary int NULL,
start_date datetime NULL,
comments varchar(255) NULL,
status_id int NULL);
```

Notice there is nothing in the query to extract the employee phone number. That process will come later. Also, all of the fields in the table are set to “NULL”. This means that a field does not have to be populated with data. The reason for this is to ensure the data gets populated. We will then examine the results to be sure that the process can continue.

STEP 3:

Once the temporary table is created, the 66 employee records that we want to examine are inserted into the temporary table. The insert requires the query used to identify the 66 records. The full insert query is shown below:

```
insert into emps(
select employee.eid, efn, eln, estreet, ecity, est, ezip,
   contact_id, contact, contact, pos_id, salary, start_date, comments, status_id
from Employee join EmpHistory on Employee.eid = EmpHistory.eid
          join Address on Address.eid = Employee.eid
                    join EmpContact on EmpContact.eid = Employee.eid
where contact_id = 2
group by contact
having count(*)>1
order by contact
);
```

Now do a select on all of the data in the temporary table **emps**:

```
select * from emps;
```

The query should return a total of 33 records. Once the **emps** table has data, compare that data to what is in the main database. The query below will compare the email address in both the **emps** table and the main database. It is based on a common email address. This will return 66 rows. These rows are a means of comparison so that those responsible for identifying the records to delete can make a good comparison of the data and make a good decision about the records to remove.

```
select e.eid, e.efn, e.eln, a.estreet, a.ecity, a.est, a.ezip,
   ec.contact_id, ec.contact, eh.pos_id, eh.salary, eh.start_date, eh.comments, eh.status_id
from Employee e join EmpHistory eh on e.eid = eh.eid
          join Address a on a.eid = e.eid
                    join EmpContact ec on ec.eid = e.eid
                              join emps
where ec.contact_id = 2 and ec.contact = emps.email
order by contact;
```

STEP 4:

Based on the results of the query above, here is a list of the 33 employee records that will be deleted from the database. The key to a successful clean-up is to delete the employees according to their employee id.

14	23	38	84	92	140
15	24	43	87	93	141
16	25	44	88	129	155
17	34	54	90	135	156
18	35	65	91	137	157
19		67		139	

Before removing any employee records, all of the data identified for removal gets placed in a holding table. Use the code below to create this table. It is the same code used to create the temporary table **emps**. Remember that a temporary table disappears once the user disconnects from the database. The holding table is a disjointed or non-relational table that stays in the database.

```
drop table if exists DelEmps;
create table if not exists DelEmps(
eid int NULL,
efn varchar(45) NULL,
eln varchar(45) NULL,
estreet varchar(45) NULL,
ecity varchar(45) NULL,
est char(2) NULL,
ezip char(5) NULL,
contact_id int NULL,
email varchar(255) NULL,
phone varchar(255) NULL,
pos_id int NULL,
salary int NULL,
start_date datetime NULL,
comments varchar(255) NULL,
status_id int NULL);
```

The tables and fields required to query to support holding the data for each employee record designated for removal are shown below:

Employee	Address	EmpHistory	EmpContact
eid **(PK)**	eaid **(PK)**	eid **(PK)** **(FK)**	eid **(PK)** **(FK)**
efn	estreet	pos_id **(PK)** **(FK)**	contact_id **(PK)** **(FK)**
eln	ecity	start_date **(PK)**	contact
	est	salary	
	ezip	comments	
	eid **(FK)**	status_id	

The queries used in this process are utilizing these fields so that the data required to archive gets placed in the holding table. The one exception is phone data. There is a field in the temporary table and in the holding table named **phone**. It is set to a datatype of varchar(255). It is temporarily populated with the email address. In the final steps of this process, it will be updated with the employees' phone number. Below is the code to insert the records to delete into the holding table **DelEmp**.

```
insert into DelEmps(
select e.eid, efn, eln, estreet, ecity, est, ezip,
   contact_id, contact, contact, pos_id, salary, start_date, comments, status_id
from Employee e join EmpHistory on e.eid = EmpHistory.eid
         join Address on Address.eid = e.eid
                  join EmpContact on EmpContact.eid = e.eid
where contact_id = 2 and (e.eid=129 or e.eid=155 or e.eid=91 or e.eid=157
         or e.eid=137 or e.eid=141 or e.eid=19 or e.eid=16 or e.eid=84
         or e.eid=15 or e.eid=92 or e.eid=88 or e.eid=65 or e.eid=90
         or e.eid=139 or e.eid=135 or e.eid=43 or e.eid=14 or e.eid=35
         or e.eid=67 or e.eid=25 or e.eid=23 or e.eid=156 or e.eid=87
         or e.eid=24 or e.eid=44 or e.eid=34 or e.eid=17 or e.eid=140
         or e.eid=18 or e.eid=93 or e.eid=38 or e.eid=54)
);

select * from DelEmps;
```

STEP 5:

The data in the holding table still needs the phone contact for each of the records in the table. In this step, the code retrieves the phone contact and then updates the email addresses in the phone field with the correct phone contact information from the database. Create a table named **PhoneEmps** as shown below.

```
drop table if exists PhoneEmps;
create table if not exists PhoneEmps(
eid int NULL,
contact varchar(255) NULL
);
```

Next, insert the phone data from the employee records designated for removal using this code below:

```
/*
This is extremely inefficient code. The challenge is to first complete the process
of correcting the problem of duplicate employee records. Then take the insert
code below and make it more efficient. The update code that follows give you an
idea of how to go improve this code.

insert into PhoneEmps(
select e.eid, contact
from Employee e join EmpContact ec on ec.eid = e.eid
where contact_id = 1 and (e.eid=129 or e.eid=155 or e.eid=91 or e.eid=157
        or e.eid=137 or e.eid=141 or e.eid=19 or e.eid=16 or e.eid=84
        or e.eid=15 or e.eid=92 or e.eid=88 or e.eid=65 or e.eid=90
        or e.eid=139 or e.eid=135 or e.eid=43 or e.eid=14 or e.eid=35
        or e.eid=67 or e.eid=25 or e.eid=23 or e.eid=156 or e.eid=87
        or e.eid=24 or e.eid=44 or e.eid=34 or e.eid=17 or e.eid=140
        or e.eid=18 or e.eid=93 or e.eid=38 or e.eid=54)
);
*/
```

```
insert into PhoneEmps(
select e.eid, contact
from Employee e join EmpContact ec on ec.eid = e.eid
where contact_id = 1 and e.eid in (select eid from DelEmps));
```

```
select * from PhoneEmps;
```

With the data in the **PhoneEmps** table, update the phone content in the **DelEmps** table. The code below achieves this:

```
update DelEmps join PhoneEmps
  on DelEmps.eid = PhoneEmps.eid
set DelEmps.phone = PhoneEmps.contact;
```

```
select * from DelEmps;
```

This completes the process to save the duplicate employee records. The next step will delete these records from the database. After this, it is important to create a process that will capture any potential duplicate record for review and will prevent it from entering the database until confirmation of its legitimacy.

Before deleting these records, all of the foreign key constraints require an update. All of the child tables have a foreign key constraint where the delete and update tasks are set to **NO ACTION**. In order to modify the affect child tables, all of the constraints will get deleted and then new constraints added with the **ON DELETE** and the **ON UPDATE** are set to **CASCADE**. The affected child tables in this process are:

- **EmpHistory**
- **EmpContact**
- **Address**

The code that follows modifies all of the affected tables:

```
alter table Address
drop constraint `fk_Address_Employee`;

alter table Address
add constraint `fk_Address_Employee`
  FOREIGN KEY (`eid`)
```

```
    REFERENCES `Employee` (`eid`)
    ON DELETE CASCADE
    ON UPDATE CASCADE;

alter table EmpContact
drop CONSTRAINT `fk_Employee_has_Contact_Employee1`;

alter table EmpContact
add constraint `fk_Employee_has_Contact_Employee1`
    FOREIGN KEY (`eid`)
    REFERENCES `Employee` (`eid`)
    ON DELETE CASCADE
    ON UPDATE CASCADE;

alter table EmpContact
drop CONSTRAINT `fk_Employee_has_Contact_Employee1`;

alter table EmpContact
add constraint `fk_Employee_has_Contact_Employee1`
    FOREIGN KEY (`eid`)
    REFERENCES `Employee` (`eid`)
    ON DELETE CASCADE
    ON UPDATE CASCADE;

alter table EmpHistory
drop constraint `5`;

alter table EmpHistory
add CONSTRAINT `fk_EmpHist_1`
    FOREIGN KEY (`eid`)
    REFERENCES `Employee` (`eid`)
    ON DELETE CASCADE
    ON UPDATE CASCADE
```

The child tables are set to cascade the delete that gets initiated from the **Employee** table. To delete the duplicate records identified earlier use the code below:

```
delete from employee
where eid in (select eid from DelEmps);
```

The code below will help confirm that the duplicate records identified for deletion have been deleted and the process is complete. Next, salary adjustments for the 129 employees who require it.

```
select * from Employee;
select * from EmpContact;
select * from EmpHistory;
select * from Address;
```