Lab 4. Inserting, Updating, Deleting

The past few labs have focused on basic query techniques, all centered around the task of getting data out of a database. This lab turns the tables, and focuses on the following three topic areas:

- Inserting new records into your database
- Updating existing records
- Deleting records that you no longer want

For ease in finding them when you need them, recipes in this lab have been grouped by topic: all the insertion recipes come first, followed by the update recipes, and finally recipes for deleting data.

Inserting is usually a straightforward task. It begins with the simple problem of inserting a single row. Many times, however, it is more efficient to use a set-based approach to create new rows. To that end, you'll also find techniques for inserting many rows at a time.

Likewise, updating and deleting start out as simple tasks. You can update one record, and you can delete one record. But you can also update whole sets of records at once, and in very powerful ways. And there are many handy ways to delete records. For example, you can delete rows in one table depending on whether or not they exist in another table.

SQL even has a way, a relatively new addition to the standard, by which you can insert, update, and delete all at once. That may not sound like too useful a thing now, but the MERGE statement represents a very powerful way to bring a database table into sync with an external source of data (such as a flat file feed from a remote system). Check out Section in this lab for details.

4.1. Inserting a New Record

PROBLEM

You want to insert a new record into a table. For example, you want to insert a new record into the DEPT table. The value for DEPTNO should be 50, DNAME should be "PROGRAMMING", and LOC should be "BALTIMORE".

SOLUTION

Use the INSERT statement with the VALUES clause to insert one row at a time:

```
insert into dept (deptno,dname,loc)
values (50,'PROGRAMMING','BALTIMORE')
```

For DB2 and MySQL you have the option of inserting one row at a time or multiple rows at a time by including multiple VALUES lists:

```
/* multi row insert */
insert into dept (deptno,dname,loc)
values (1,'A','B'),
       (2,'B','C')
```

DISCUSSION
The INSERT statement allows you to create new rows in database tables. The syntax for inserting a single row is consistent across all database brands.

As a shortcut, you can omit the column list in an INSERT statement:

```
insert into dept
values (50,'PROGRAMMING','BALTIMORE')
```

However, if you do not list your target columns, you must insert into all of the columns in the table, and be mindful of the order of the values in the VALUES list; you must supply values in the same order in which the database displays columns in response to a SELECT * query.

4.2. Inserting Default Values

PROBLEM

A table can be defined to take default values for specific columns. You want to insert a row of default values without having to specify those values. Consider the following table:

```
create table D (id integer default 0)
```

You want to insert zero without explicitly specifying zero in the values list of an INSERT statement. You want to explicitly insert the default, whatever that default is.

SOLUTION

All brands support use of the DEFAULT keyword as a way of explicitly specifying the default value for a column. Some brands provide additional ways to solve the problem.

The following example illustrates the use of the DEFAULT keyword:

```
insert into D values (default)
```

You may also explicitly specify the column name, which you'll need to do anytime you are not inserting into all columns of a table:

```
insert into D (id) values (default)
```

Oracle8i Database and prior versions do not support the DEFAULT keyword. Prior to Oracle9i Database, there was no way to explicitly insert a default column value.

MySQL allows you to specify an empty values list if all columns have a default value defined:

```
insert into D values ()
```

In this case, all columns will be set to their default values.

PostgreSQL and SQL Server support a DEFAULT VALUES clause:

```
insert into D default values
```

The DEFAULT VALUES clause causes all columns to take on their default values.

DISCUSSION

The DEFAULT keyword in the values list will insert the value that was specified as the default for a particular column during table creation. The keyword is available for all DBMSs.

MySQL, PostgreSQL, and SQL Server users have another option available if all columns in the table are defined with a default value (as table D is in this case). You may use an empty VALUES list (MySQL) or specify the DEFAULT VALUES clause (PostgreSQL and SQL Server) to create a new row with all default values; otherwise, you need to specify DEFAULT for each column in the table.

For tables with a mix of default and non-default columns, inserting default values for a column is as easy as excluding the column from the insert list; you do not need to use the DEFAULT keyword. Say that table D had an additional column that was not defined with a default value:

```
create table D (id integer default 0, foo varchar(10))
```

You can insert a default for ID by listing only FOO in the insert list:

```
insert into D (name) values ('Bar')
```

This statement will result in a row in which ID is 0 and FOO is “Bar”. ID takes on its default value because no other value is specified.

4.3. Overriding a Default Value with NULL

PROBLEM

You are inserting into a column having a default value, and you wish to override that default value by setting the column to NULL. Consider the following table:

```
create table D (id integer default 0, foo VARCHAR(10))
```

You wish to insert a row with a NULL value for ID.

SOLUTION

You can explicitly specify NULL in your values list:

```
insert into d (id, foo) values (null, 'Brighten')
```

DISCUSSION

Not everyone realizes that you can explicitly specify NULL in the values list of an INSERT statement. Typically, when you do not wish to specify a value for a column, you leave that column out of your column and values lists:

```
insert into d (foo) values ('Brighten')
```

Here, no value for ID is specified. Many would expect the column to taken on the null value, but, alas, a default value was specified at table creation time, so the result of the preceding INSERT is that ID takes on the value 0 (the default). By specifying NULL as the value for a column, you can set the column to NULL despite any default value.

4.4. Copying Rows from One Table into Another

PROBLEM

You want to copy rows from one table to another by using a query. The query may be complex or simple, but ultimately you want the result to be inserted into another table. For example, you want to

copy rows from the DEPT table to the DEPT_EAST table. The DEPT_EAST table has already been created with the same structure (same columns and data types) as DEPT and is currently empty.

SOLUTION

Use the INSERT statement followed by a query to produce the rows you want:

```
insert into dept_east (deptno,dname,loc)
select deptno,dname,loc
  from dept
 where loc in ( 'NEW YORK','BOSTON' )
```

DISCUSSION

Simply follow the INSERT statement with a query that returns the desired rows. If you want to copy all rows from the source table, exclude the WHERE clause from the query. Like a regular insert, you do not have to explicitly specify which columns you are inserting into. But if you do not specify your target columns, you must insert into all of the table's columns, and you must be mindful of the order of the values in the SELECT list as described earlier in "Inserting a New Record."

4.5. Copying a Table Definition

PROBLEM

You want to create a new table having the same set of columns as an existing table. For example, you want to create a copy of the DEPT table and call it DEPT_2. You do not want to copy the rows, only the column structure of the table.

SOLUTION

DB2
Use the LIKE clause with the CREATE TABLE command:

```
create table dept_2 like dept
```

Oracle, MySQL, and PostgreSQL
Use the CREATE TABLE command with a subquery that returns no rows:

```
1 create table dept_2
2 as
3 select *
4   from dept
5  where 1 = 0
```

SQL Server
Use the INTO clause with a subquery that returns no rows:

```
1 select *
2   into dept_2
3   from dept
4  where 1 = 0
```

DISCUSSION

DB2
DB2's CREATE TABLE…LIKE command allows you to easily use one table as the pattern for creating another. Simply specify your pattern table's name following the LIKE keyword.

Oracle, MySQL, and PostgreSQL
When using Create Table As Select (CTAS), all rows from your query will be used to populate the new table you are creating unless you specify a false condition in the WHERE clause. In the solution provided, the expression "1 = 0" in the WHERE clause of the query causes no rows to be returned. Thus the result of the CTAS statement is an empty table based on the columns in the SELECT clause of the query.

SQL Server
When using INTO to copy a table, all rows from your query will be used to populate the new table you are creating unless you specify a false condition in the WHERE clause of your query. In the solution provided, the expression "1 = 0" in the predicate of the query causes no rows to be returned. The result is an empty table based on the columns in the SELECT clause of the query.

4.6. Inserting into Multiple Tables at Once

PROBLEM

You want to take rows returned by a query and insert those rows into multiple target tables. For example, you want to insert rows from DEPT into tables DEPT_EAST, DEPT_WEST, and DEPT_MID. All three tables have the same structure (same columns and data types) as DEPT and are currently empty.

SOLUTION

The solution is to insert the result of a query into the target tables. The difference from “Copying Rows from One Table into Another” is that for this problem you have multiple target tables.

Oracle

Use either the INSERT ALL or INSERT FIRST statement. Both share the same syntax except for the choice between the ALL and FIRST keywords. The following statement uses INSERT ALL to cause all possible target tables to be considered:

```
    insert all
      when loc in ('NEW YORK','BOSTON') then
into dept_east (deptno,dname,loc) values (deptno,dname,loc)
     when loc = 'CHICAGO' then
       into dept_mid (deptno,dname,loc) values (deptno,dname,oc)
     else
       into dept_west (deptno,dname,loc) values (deptno,dname,oc)
     select deptno,dname,loc
       from dept
```

DB2

Insert into an inline view that performs a UNION ALL on the tables to be inserted. You must also be sure to place constraints on the tables that will ensure each row goes into the correct table:

```
create table dept_east
( deptno integer,
  dname  varchar(10),
  loc    varchar(10) check (loc in ('NEW YORK','BOSTON')))

create table dept_mid
( deptno integer,
  dname  varchar(10),
  loc    varchar(10) check (loc = 'CHICAGO'))

create table dept_west
( deptno integer,
  dname  varchar(10),
  loc    varchar(10) check (loc = 'DALLAS'))

  insert into (
    select * from dept_west union all
    select * from dept_east union all
    select * from dept_mid
  ) select * from dept
```

MySQL, PostgreSQL, and SQL Server
As of the time of this writing, these vendors do not support multi-table inserts.

DISCUSSION

Oracle
Oracle's multi-table insert uses WHEN-THEN-ELSE clauses to evaluate the rows from the nested SELECT and insert them accordingly. In this recipe's example, INSERT ALL and INSERT FIRST would produce the same result, but there is a difference between the two. INSERT FIRST will break out of the WHEN-THEN-ELSE evaluation as soon as it encounters a condition evaluating to true; INSERT ALL will evaluate all conditions even if prior tests evaluate to true. Thus, you can use INSERT ALL to insert the same row into more than one table.

DB2
My DB2 solution is a bit of a hack. It requires that the tables to be inserted into have constraints defined to ensure that each row evaluated from the subquery will go into the correct table. The technique is to insert into a view that is defined as the UNION ALL of the tables. If the check constraints are not unique amongst the tables in the INSERT (i.e., multiple tables have the same check constraint), the INSERT statement will not know where to put the rows and it will fail.

MySQL, PostgreSQL, and SQL Server
As of the time of this writing, only Oracle and DB2 currently provide mechanisms to insert rows returned by a query into one or more of several tables within the same statement.

4.7. Blocking Inserts to Certain Columns

PROBLEM

You wish to prevent users, or an errant software application, from inserting values into certain table columns. For example, you wish to allow a program to insert into EMP, but only into the EMPNO, ENAME, and JOB columns.

SOLUTION

Create a view on the table exposing only those columns you wish to expose. Then force all inserts to go through that view.

For example, to create a view exposing the three columns in EMP:

```
create view new_emps as
select empno, ename, job
  from emp
```

Grant access to this view to those users and programs allowed to populate only the three fields in the view. Do not grant those users insert access to the EMP table. Users may then create new EMP records by inserting into the NEW_EMPS view, but they will not be able to provide values for columns other than the three that are specified in the view definition.

DISCUSSION

When you insert into a simple view such as in the solution, your database server will translate that insert into the underlying table. For example, the following insert:

```
insert into new_emps
   (empno ename, job)
   values (1, 'Jonathan', 'Editor')
```

will be translated behind the scenes into:

```
insert into emp
   (empno ename, job)
   values (1, 'Jonathan', 'Editor')
```

It is also possible, but perhaps less useful, to insert into an inline view (currently only supported by Oracle):

```
insert into
  (select empno, ename, job
     from emp)
values (1, 'Jonathan', 'Editor')
```

View insertion is a complex topic. The rules become very complicated very quickly for all but the simplest of views. If you plan to make use of the ability to insert into views, it is imperative that you consult and fully understand your vendor documentation on the matter.

4.8. Modifying Records in a Table

PROBLEM

You want to modify values for some or all rows in a table. For example, you might want to increase the salaries of everyone in department 20 by 10%. The following result set shows the DEPTNO, ENAME, and SAL for employees in that department:

```
select deptno,ename,sal
  from emp
 where deptno = 20
 order by 1,3

DEPTNO ENAME             SAL
------ ---------- ----------
    20 SMITH             800
    20 ADAMS            1100
    20 JONES            2975
    20 SCOTT            3000
    20 FORD             3000
```

You want to bump all the SAL values by 10%.

SOLUTION

Use the UPDATE statement to modify existing rows in a database table. For example:

```
update emp
   set sal = sal*1.10
 where deptno = 20
```

DISCUSSION

Use the UPDATE statement along with a WHERE clause to specify which rows to update; if you exclude a WHERE clause, then all rows are updated. The expression SAL*1.10 in this solution returns the salary increased by 10%.

When preparing for a mass update, you may wish to preview the results. You can do that by issuing a SELECT statement that includes the expressions you plan to put into your SET clauses. The following SELECT shows the result of a 10% salary increase:

```
select deptno,
       ename,
       sal      as orig_sal,
       sal*.10  as amt_to_add,
       sal*1.10 as new_sal
  from emp
 where deptno=20
 order by 1,5

DEPTNO ENAME  ORIG_SAL AMT_TO_ADD  NEW_SAL
------ ------ -------- ----------  -------
    20 SMITH       800          80      880
    20 ADAMS      1100         110     1210
    20 JONES      2975         298     3273
    20 SCOTT      3000         300     3300
    20 FORD       3000         300     3300
```

The salary increase is broken down into two columns: one to show the increase over the old salary, and the other to show the new salary.

4.9. Updating when Corresponding Rows Exist

PROBLEM

You want to update rows in one table when corresponding rows exist in another. For example, if an employee appears in table EMP_BONUS, you want to increase that employee's salary (in table EMP) by 20 percent. The following result set represents the data currently in table EMP_BONUS:

```
select empno, ename
  from emp_bonus

     EMPNO ENAME
---------- ---------
      7369 SMITH
      7900 JAMES
      7934 MILLER
```

SOLUTION

Use a subquery in your UPDATE statement's WHERE clause to find employees in table EMP that are also in table EMP_BONUS. Your UPDATE will then act only on those rows, enabling you to increase their salary by 20 percent:

```
update emp
   set sal=sal*1.20
 where empno in ( select empno from emp_bonus )
```

DISCUSSION

The results from the subquery represent the rows that will be updated in table EMP. The IN predicate tests values of EMPNO from the EMP table to see whether they are in the list of EMPNO values returned by the subquery. When they are, the corresponding SAL values are updated.

Alternatively, you can use EXISTS instead of IN:

```
update emp
   set sal = sal*1.20
 where exists ( select null
                  from emp_bonus
                 where emp.empno=emp_bonus.empno )
```

You may be surprised to see NULL in the SELECT list of the EXISTS subquery. Fear not, that NULL does not have an adverse effect on the update. In my opinion it increases readability as it reinforces the fact that, unlike the solution using a subquery with an IN operator, what will drive the update (i.e., which rows will be updated) will be controlled by the WHERE clause of the subquery, not the values returned as a result of the subquery's SELECT list.

4.10. Updating with Values from Another Table

PROBLEM

You wish to update rows in one table using values from another. For example, you have a table called NEW_SAL, which holds the new salaries for certain employees. The contents of table NEW_SAL are:

```
select *
  from new_sal

DEPTNO        SAL
------ ----------
    10       4000
```

Column DEPTNO is the primary key of table NEW_SAL. You want to update the salaries and commission of certain employees in table EMP using values table NEW_SAL if there is a match between EMP.DEPTNO and NEW_SAL.DEPTNO, update EMP.SAL to NEW_SAL.SAL, and update EMP.COMM to 50% of NEW_SAL.SAL. The rows in EMP are as follows:

```
select deptno,ename,sal,comm
  from emp
 order by 1

DEPTNO ENAME              SAL       COMM
------ ---------- ---------- ----------
    10 CLARK            2450
    10 KING             5000
    10 MILLER           1300
    20 SMITH             800
    20 ADAMS            1100
    20 FORD             3000
    20 SCOTT            3000
    20 JONES            2975
    30 ALLEN            1600          300
    30 BLAKE            2850
    30 MARTIN           1250         1400
    30 JAMES             950
    30 TURNER           1500            0
    30 WARD             1250          500
```

SOLUTION

Use a join between NEW_SAL and EMP to find and return the new COMM values to the UPDATE statement. It is quite common for updates such as this one to be performed via correlated subquery. Another technique involves creating a view (traditional or inline, depending on what your database supports), then updating that view.

DB2

Use a correlated subquery to set new SAL and COMM values in EMP. Also use a correlated subquery to identify which rows from EMP should be updated:

```
update emp e set (e.sal,e.comm) = (select ns.sal, ns.sal/2
                                     from new_sal ns
                                    where ns.deptno=e.deptno)
 where exists ( select *
                  from new_sal ns
                 where ns.deptno = e.deptno )
```

MySQL

Include both EMP and NEW_SAL in the UPDATE clause of the UPDATE statement and join in the WHERE clause:

```
update emp e, new_sal ns
set e.sal=ns.sal,
e.comm=ns.sal/2
where e.deptno=ns.deptno
```

Oracle

The method for the DB2 solution will certainly work for Oracle, but as an alternative, you can update an inline view:

```
update (
 select e.sal as emp_sal, e.comm as emp_comm,
        ns.sal as ns_sal, ns.sal/2 as ns_comm
   from emp e, new_sal ns
  where e.deptno = ns.deptno
) set emp_sal = ns_sal, emp_comm = ns_comm
```

PostgreSQL

The method used for the DB2 solution will work for PostgreSQL, but as an alternative you can (quite conveniently) join directly in the UPDATE statement:

```
update emp
   set sal = ns.sal,
       comm = ns.sal/2
  from new_sal ns
 where ns.deptno = emp.deptno
```

SQL Server

The method used for the DB2 solution will work for SQL Server, but as an alternative you can (similarly to the PostgreSQL solution) join directly in the UPDATE statement:

```
update e
   set e.sal  = ns.sal,
       e.comm = ns.sal/2
  from emp e,
       new_sal ns
 where ns.deptno = e.deptno
```

DISCUSSION

Before discussing the different solutions, I'd like to mention something important regarding updates that use queries to supply new values. A WHERE clause in the subquery of a correlated update is not the same as the WHERE clause of the table being updated. If you look at the UPDATE statement in the "Problem" section, the join on DEPTNO between EMP and NEW_SAL is done and returns rows to

the SET clause of the UPDATE statement. For employees in DEPTNO 10, valid values are returned because there is a match DEPTNO in table NEW_SAL. But what about employees in the other departments? NEW_SAL does not have any other departments, so the SAL and COMM for employees in DEPTNOs 20 and 30 are set to NULL. Unless you are doing so via LIMIT or TOP or whatever mechanism your vendor supplies for limiting the number of rows returned in a result set, the only way to restrict rows from a table in SQL is to use a WHERE clause. To correctly perform this UPDATE, use a WHERE clause on the table being updated along with a WHERE clause in the correlated subquery.

DB2

To ensure you do not update every row in table EMP, remember to include a correlated subquery in the WHERE clause of the UPDATE. Performing the join (the correlated subquery) in the SET clause is not enough. By using a WHERE clause in the UPDATE, you ensure that only rows in EMP that match on DEPTNO to table NEW_SAL are updated. This holds true for all RDBMSs.

Oracle

In the Oracle solution using the update join view, you are using equi-joins to determine which rows will be updated. You can confirm which rows are being updated by executing the query independently. To be able to successfully use this type of UPDATE, you must first understand the concept of key-preservation. The DEPTNO column of the table NEW_SAL is the primary key of that table, thus its values are unique within the table. When joining between EMP and NEW_SAL, however, NEW_SAL.DEPTNO is not unique in the result set, as can be seen below:

```
select e.empno, e.deptno e_dept, ns.sal, ns.deptno ns_deptno
  from emp e, new_sal ns
 where e.deptno = ns.deptno

EMPNO     E_DEPT        SAL  NS_DEPTNO
----- ---------- ---------- ----------
 7782         10       4000         10
 7839         10       4000         10
 7934         10       4000         10
```

To enable Oracle to update this join, one of the tables must be key-preserved, meaning that if its values are not unique in the result set, it should at least be unique in the table it comes from. In this case NEW_SAL has a primary key on DEPTNO, which makes it unique in the table. Because it is unique in its table, it may appear multiple times in the result set and will still be considered key-preserved, thus allowing the update to complete successfully.

PostgreSQL, SQL Server, and MySQL

The syntax is a bit different between these platforms, but the technique is the same. Being able to join directly in the UPDATE statement is extremely convenient. Since you specify which table to

update (the table listed after the UPDATE keyword) there's no confusion as to which table's rows are modified. Additionally, because you are using joins in the update (since there is an explicit WHERE clause), you can avoid some of the pitfalls when coding correlated subquery updates; in particular, if you missed a join here, it would be very obvious you'd have a problem.

4.11. Merging Records

PROBLEM

You want to conditionally insert, update, or delete records in a table depending on whether or not corresponding records exist. (If a record exists, then update; if not,then insert; if after updating a row fails to meet a certain condition, delete it.) For example, you want to modify table EMP_COMMISSION such that:

If any employee in EMP_COMMISSION also exists in table EMP, then update their commission (COMM) to 1000.

For all employees who will potentially have their COMM updated to 1000, if their SAL is less than 2000, delete them (they should not be exist in EMP_COMMISSION).

Otherwise, insert the EMPNO, ENAME, and DEPTNO values from table EMP into table EMP_COMMISSION.

Essentially, you wish to execute either an UPDATE or an INSERT depending on whether a given row from EMP has a match in EMP_COMMISSION. Then you wish to execute a DELETE if the result of an UPDATE causes a commission that's too high.

The following rows are currently in tables EMP and EMP_COMMISSION, respectively:

```
select deptno,empno,ename,comm
  from emp
 order by 1

DEPTNO      EMPNO ENAME        COMM
------ ---------- ------ ----------
    10       7782  CLARK
    10       7839  KING
    10       7934  MILLER
    20       7369  SMITH
    20       7876  ADAMS
    20       7902  FORD
    20       7788  SCOTT
    20       7566  JONES
    30       7499  ALLEN        300
    30       7698  BLAKE
    30       7654  MARTIN      1400
    30       7900  JAMES
    30       7844  TURNER         0
    30       7521  WARD         500


select deptno,empno,ename,comm
  from emp_commission
 order by 1

    DEPTNO      EMPNO ENAME            COMM
---------- ---------- ---------- ----------
        10       7782 CLARK
        10       7839 KING
        10       7934 MILLER
```

SOLUTION

Oracle is currently the only RDBMS with a statement designed to solve this problem. That statement is the MERGE statement, and it can perform either an UPDATE or an INSERT, as needed. For example:

```
  merge into emp_commission ec
  using (select * from emp) emp
     on (ec.empno=emp.empno)
   when matched then
        update set ec.comm = 1000
        delete where (sal < 2000)
   when not matched then
        insert (ec.empno,ec.ename,ec.deptno,ec.comm)
        values (emp.empno,emp.ename,emp.deptno,emp.comm)
```

DISCUSSION

The join on line 3 of the solution determines what rows already exist and will be updated. The join is between EMP_COMMISSION (aliased as EC) and the subquery (aliased as emp). When the join succeeds, the two rows are considered “matched” and the UPDATE specified in the WHEN MATCHED clause is executed. Otherwise, no match is found and the INSERT in WHEN NOT MATCHED is executed. Thus, rows from table EMP that do not have corresponding rows based on EMPNO in table EMP_COMMISSION will be inserted into EMP_COMMISSION. Of all the employees in table EMP only those in DEPTNO 10 should have their COMM updated in EMP_COMMISSION, while the rest of the employees are inserted. Additionally, since MILLER is in DEPTNO 10 he is a candidate to have his COMM updated, but because his SAL is less than 2000 it is deleted from EMP_COMMISSION.

4.12. Deleting All Records from a Table

PROBLEM

You want to delete all the records from a table.

SOLUTION

Use the DELETE command to delete records from a table. For example, to delete all records from EMP:

```
delete from emp
```

DISCUSSION

When using the DELETE command without a WHERE clause, you will delete all rows from the table specified.

4.13. Deleting Specific Records

PROBLEM

You wish to delete records meeting a specific criterion from a table.

SOLUTION

Use the DELETE command with a WHERE clause specifying which rows to delete. For example, to delete all employees in department 10:

```
delete from emp where deptno = 10
```

DISCUSSION

By using a WHERE clause with the DELETE command, you can delete a subset of rows in a table rather than all the rows.

4.14. Deleting a Single Record

PROBLEM

You wish to delete a single record from a table.

SOLUTION

This is a special case of "Deleting Specific Records." The key is to ensure that your selection criterion is narrow enough to specify only the one record that you wish to delete. Often you will want to delete based on the primary key. For example, to delete employee CLARK (EMPNO 7782):

```
delete from emp where empno = 7782
```

DISCUSSION

Deleting is always about identifying the rows to be deleted, and the impact of a DELETE always comes down to its WHERE clause. Omit the WHERE clause and the scope of a DELETE is the entire table. By writing conditions in the WHERE clause, you can narrow the scope to a group of records, or to a single record. When deleting a single record, you should typically be identifying that record based on its primary key or on one of its unique keys.

WARNING

If your deletion criterion is based on a primary or unique key, then you can be sure of deleting only one record. (This is because your RDBMS will not allow two rows to contain the same primary or unique key values.) Otherwise, you may want to check first, to be sure you aren't about to inadvertently delete more records than you intend.

4.15. Deleting Referential Integrity Violations

PROBLEM

You wish to delete records from a table when those records refer to nonexistent records in some other table. Example: some employees are assigned to departments that do not exist. You wish to delete those employees.

SOLUTION

Use the NOT EXISTS predicate with a subquery to test the validity of department numbers:

```
delete from emp
 where not exists (
   select * from dept
    where dept.deptno = emp.deptno
)
```

Alternatively, you can write the query using a NOT IN predicate:

```
delete from emp
where deptno not in (select deptno from dept)
```

DISCUSSION

Deleting is really all about selecting: the real work lies in writing WHERE clause conditions to correctly describe those records that you wish to delete.

The NOT EXISTS solution uses a correlated subquery to test for the existence of a record in DEPT having a DEPTNO matching that in a given EMP record. If such a record exists, then the EMP record is retained. Otherwise, it is deleted. Each EMP record is checked in this manner.

The IN solution uses a subquery to retrieve a list of valid department numbers. DEPTNOs from each EMP record are then checked against that list. When an EMP record is found with a DEPTNO not in the list, the EMP record is deleted.

4.16. Deleting Duplicate Records

PROBLEM

You want to delete duplicate records from a table. Consider the following table:

```
create table dupes (id integer, name varchar(10))
insert into dupes values (1, 'NAPOLEON')
insert into dupes values (2, 'DYNAMITE')
insert into dupes values (3, 'DYNAMITE')
insert into dupes values (4, 'SHE SELLS')
insert into dupes values (5, 'SEA SHELLS')
insert into dupes values (6, 'SEA SHELLS')
insert into dupes values (7, 'SEA SHELLS')


select * from dupes order by 1

        ID NAME
---------- ----------
         1 NAPOLEON
         2 DYNAMITE
         3 DYNAMITE
         4 SHE SELLS
         5 SEA SHELLS
         6 SEA SHELLS
         7 SEA SHELLS
```

For each group of duplicate names, such as “SEA SHELLS”, you wish to arbitrarily retain one ID and delete the rest. In the case of “SEA SHELLS” you don’t care whether you delete 5 and 6, or 5 and 7, or 6 and 7, but in the end you want just one record for “SEA SHELLS”.

SOLUTION

Use a subquery with an aggregate function such as MIN to arbitrarily choose the ID to retain (in this case only the NAME with the smallest value for ID is not deleted):

```
delete from dupes
 where id not in ( select min(id)
                     from dupes
                    group by name )
```

For MySQL users you will need slightly different syntax because you cannot reference the same table twice in a delete (as of the time of this writing):

```
delete from dupes
 where id not in
       (select min(id)
 from (select id,name from dupes) tmp
        group by name)
```

DISCUSSION

The first thing to do when deleting duplicates is to define exactly what it means for two rows to be considered “duplicates” of each other. For my example in this recipe, the definition of “duplicate” is that two records contain the same value in their NAME column. Having that definition in place, you can look to some other column to discriminate among each set of duplicates, to identify those records to retain. It’s best if this discriminating column (or columns) is a primary key. I used the ID column, which is a good choice because no two records have the same ID.

The key to the solution is that you group by the values that are duplicated (by NAME in this case), and then use an aggregate function to pick off just one key value to retain. The subquery in the “Solution” example will return the smallest ID for each NAME, which represents the row you will not delete:

```
select min(id)
  from dupes
 group by name

    MIN(ID)
-----------
          2
          1
          5
          4
```

The DELETE then deletes any ID in the table that is not returned by the subquery (in this case IDs 3, 6, and 7). If you are having trouble seeing how this works, run the subquery first and include the NAME in the SELECT list:

```
select name, min(id)
  from dupes
 group by name

NAME          MIN(ID)
---------- ----------
DYNAMITE            2
NAPOLEON            1
SEA SHELLS          5
SHE SELLS           4
```

The rows returned by the subquery represent those to be retained. The NOT IN predicate in the DELETE statement causes all other rows to be deleted.

4.17. Deleting Records Referenced from Another Table

PROBLEM

You want to delete records from one table when those records are referenced from some other table. Consider the following table, named DEPT_ACCIDENTS, which contains one row for each accident that occurs in a manufacturing business. Each row records the department in which an accident occurred and also the type of accident.

```
create table dept_accidents
( deptno         integer,
  accident_name  varchar(20) )

insert into dept_accidents values (10,'BROKEN FOOT')
insert into dept_accidents values (10,'FLESH WOUND')
insert into dept_accidents values (20,'FIRE')
insert into dept_accidents values (20,'FIRE')
insert into dept_accidents values (20,'FLOOD')
insert into dept_accidents values (30,'BRUISED GLUTE')

select * from dept_accidents

    DEPTNO ACCIDENT_NAME
---------- --------------------
        10 BROKEN FOOT
        10 FLESH WOUND
        20 FIRE
        20 FIRE
        20 FLOOD
        30 BRUISED GLUTE
```

You want to delete from EMP the records for those employees working at a department that has three or more accidents.

SOLUTION

Use a subquery and the aggregate function COUNT to find the departments with three or more accidents. Then delete all employees working in those departments:

```
 delete from emp
  where deptno in ( select deptno
                      from dept_accidents
                     group by deptno
                    having count(*) >= 3 )
```

DISCUSSION

The subquery will identify which departments have three or more accidents:

```
select deptno
  from dept_accidents
 group by deptno
having count(*) >= 3

    DEPTNO
----------
        20
```

The DELETE will then delete any employees in the departments returned by the subquery (in this case, only in department 20).