Delete with joins

To delete data from a table, you use the MySQL DELETE statement. The following illustrates the syntax of the DELETE statement:

```
1 DELETE FROM table_name
2 WHERE condition;
```

In this statement:

- First, specify the table from which you delete data.
- Second, use a condition to specify which rows to delete in the WHERE clause. If the row matches the condition, it will be deleted.

Notice that the WHERE clause is optional. If you omit the WHERE clause, the DELETE statement will delete all rows in the table.

Besides deleting data from a table, the DELETE statement returns the number of rows deleted.

Delete can also be used with limit clause.

```
DELETE FROM table
LIMIT row_count;
```

It makes more sense to use order by while using limit clause.

```
DELETE FROM table_name
ORDER BY c1, c2, ...
LIMIT row_count;
```

```
DELETE FROM customers
WHERE country = 'France'
ORDER BY creditLimit
LIMIT 5;
```

For a table that has a foreign key constraint, when you delete rows from the parent table, the rows in the child table will be deleted automatically by using the ON DELETE CASCADE option.

```
CREATE TABLE rooms (
    room_no INT PRIMARY KEY AUTO_INCREMENT,
    room_name VARCHAR(255) NOT NULL,
    building_no INT NOT NULL,
    FOREIGN KEY (building_no)
```

```
        REFERENCES buildings (building_no)
        ON DELETE CASCADE
);
```

Notice that ON DELETE CASCADE works only with tables with the storage engines support foreign keys e.g., InnoDB. Some table types do not support foreign keys such as MyISAM so you should choose appropriate storage engines for the tables that you plan to use the MySQL ON DELETE CASCADE referential action.

Tips to find tables affected by MySQL ON DELETE CASCADE action

Sometimes, it is useful to know which table is affected by the MySQL ON DELETE CASCADE referential action when you delete data from a table. You can query this data from the referential_constraintsin the information_schema database as follows:

```
   SELECT  table_name

from information_schema.key_column_usage

where constraint_schema = ‘db_name’

AND delete_rule = 'CASCADE'
```

In the SQL 2003 standard there are 5 different referential actions:

1. CASCADE
2. RESTRICT
3. NO ACTION
4. SET NULL
5. SET DEFAULT

To answer the question:

1. **CASCADE**
   - ON DELETE CASCADE means that if the parent record is deleted, any child records are also deleted. This is **not** a good idea in my opinion. You should keep track of all data that's ever been in a database, although this can be done using TRIGGERs. (However, see caveat in comments below).
   - ON UPDATE CASCADE means that if the parent primary key is changed, the child value will also change to reflect that. Again in my opinion, not a great idea. If you're changing PRIMARY KEYs with any regularity (or even at all!), there is something wrong with your design. Again, see comments.
   - ON UPDATE CASCADE ON DELETE CASCADE means that if you UPDATE **OR** DELETE the parent, the change is cascaded to the child. This is the equivalent of ANDing the outcomes of first two statements.
2. **RESTRICT**

- RESTRICT means that any attempt to delete and/or update the parent will fail throwing an error. This is the default behaviour in the event that a referential action is not explicitly specified.
  For an ON DELETE or ON UPDATE that is not specified, the default action is always RESTRICT`.

3. **NO ACTION**
   - NO ACTION: From the manual. A keyword from standard SQL. In MySQL, equivalent to RESTRICT. The MySQL Server rejects the delete or update operation for the parent table if there is a related foreign key value in the referenced table. Some database systems have deferred checks, and NO ACTION is a deferred check. In MySQL, foreign key constraints are checked immediately, so NO ACTION is the same as RESTRICT.
4. **SET NULL**
   - SET NULL - again from the manual. Delete or update the row from the parent table, and set the foreign key column or columns in the child table to NULL. This is not the best of ideas IMHO, primarily because there is no way of "time-travelling" - i.e. looking back into the child tables and associating records with NULLs with the relevant parent record - either CASCADEor use TRIGGERs to populate logging tables to track changes (but, see comments).
5. **SET DEFAULT**
   - SET DEFAULT. Yet another (potentially very useful) part of the SQL standard that MySQL hasn't bothered implementing! Allows the developer to specify a value to which to set the foreign key column(s) on an UPDATE or a DELETE. InnoDB and NDB will reject table definitions with a SET DEFAULT clause.

Using Delete with joins

To delete rows from both T1 and T2 tables that meet a specified condition, you use the following statement:

```
DELETE T1, T2
FROM T1
INNER JOIN T2 ON T1.key = T2.key
WHERE condition;
```

Notice that you put table names T1 and T2 between the DELETE and FROM keywords. If you omit T1 table, the DELETE statement only deletes rows in T2 table. Similarly, if you omitT2 table, the DELETE statement will delete only rows in T1 table.

```
DELETE t1,t2 FROM t1
        INNER JOIN
    t2 ON t2.ref = t1.id
WHERE
    t1.id = 1;
```

```
delete f
from friendship f , fbuser u
where f.userid = u.userid
and username = "Deepa"
```

We often use the LEFT JOIN clause in the SELECT statement to find rows in the left table that have or don't have matching rows in the right table.
We can also use the LEFT JOIN clause in the DELETE statement to delete rows in a table (left table) that does not have matching rows in another table (right table).
The following syntax illustrates how to use DELETE statement with LEFT JOIN clause to delete rows from T1 table that does not have corresponding rows in the T2 table:

```
DELETE T1
FROM T1
        LEFT JOIN
    T2 ON T1.key = T2.key
WHERE
    T2.key IS NULL;
```

Note that we only put T1 table after the DELETE keyword, not both T1 and T2 tables like we did with the INNER JOIN clause.

```
select (count(a.plan) * 50 ) as planaAmt , (sum(b.hours) * 15) as planBAmt,
(count(a.plan) * 50 )+ (sum(b.hours) * 15)

from hire a , hire b

where a.plan = 'A'

and b.plan='B'


group by a.bicycle_id , a.plan
```

```
/***/
select distinct h.bicycle_id , a_amt + b_amt
from hire h, ( select bicycle_id as bid1 , count(*) * 50 as a_amt
from hire a
where plan='A'
group by a.bicycle_id )
hire_with_planA ,
 ( select bicycle_id as bid2 ,(sum(b.hours) * 15)  as b_amt
 from hire b where plan = 'B' group by b.bicycle_id) hire_with_planB
 where h.bicycle_id = hire_with_planA.bid1
 and h.bicycle_id = hire_with_planB.bid2



/****/


select (count(a.plan) * 50 ) as planaAmt , (sum(b.hours) * 15) as planBAmt,
(count(a.plan) * 50 )+ (sum(b.hours) * 15)
from hire a , hire b
where a.plan = 'A'
and b.plan='B'


group by a.bicycle_id , a.plan


/** with sub query **/
```

```
select distinct h.bicycle_id , a_amt + b_amt
from hire h left join  ( select bicycle_id as bid1 , count(*) * 50 as a_amt
from hire a
where plan='A'
group by a.bicycle_id )
mirror_1 on (h.bicycle_id = mirror_1.bid1) left join
 ( select bicycle_id as bid2 ,(sum(b.hours) * 15)  as b_amt
 from hire b where plan = 'B' group by b.bicycle_id) mirror_2
 on( h.bicycle_id = mirror_2.bid2)
```