```
/*
 * Run a series of updates on the data
 */

/* Set the status as 'complete' for three low level tickets
 * These three low-level tickets were partitioned from the same high level
ticket
 * This is done on purpose, and will play into the next (more interesting)
update...
 */
update ticket
    set status = 'complete'
    where tid in (169, 170, 171);

-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
-- -- -- --

/*
 * Update the status for all high-level tickets to 'complete',
 * iff all the low-level tickets partitioned from said high level tickets
 * have status 'complete'.
 *
 * After the previous update, all the low-level tickets associated with the
 * high level ticket with ticket id `1` (associated low-level tickets have
ticket id's
 * 169, 170, 171), will be set to 'complete', but none others will be
altered.
 */
with ticketstatuses as (
    select ht.tid,
        t.status,
        lt.tid as lowtickettid,
        tt.status as lowticketstatus
    from highticket ht
    inner join ticket t
        on ht.tid = t.tid
    inner join lowticket lt
        on ht.tid = lt.highticketid
    inner join ticket tt
        on lt.tid = tt.tid
    order by lowtickettid
```

```
)
update ticket
    set status = 'complete'
    where tid in (
        select distinct t.tid
        from(
            select *,
                count(1) over(partition by tid) as numlowleveltickets,
                sum(case
                    when lowticketstatus = 'complete'
                        then 1
                        else 0
                    end) over(partition by tid) as
numcompletelowleveltickets
            from ticketstatuses
        ) t
        where t.numlowleveltickets = t.numcompletelowleveltickets
    );

-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
-- --

/*
 * It might be the case that an employee is on sick/maternity leave,
 * and all of his/her responsibilites need to be transferred over to another
emplyee.
 *
 * The employee on leave may be a developer - in which case low-level
tickets may be assigned to him/her -
 * or a business onwer - in which case he/she may be accountable for one or
more projects.
 *
 * Below, I'll transfer all the responsibilities of employee with person id
'2' to the employee with person id '1'.
 *
 * The trouble with this is that the responsibilites of an employee may span
across multiple tables -
 * for example, a ticket may be assigned to that employee, that employee may
be an accountable business owner.
 * This adds complexity becasue the column that needs to be changed in each
of the tables may be different -
```

```
 * for example in the table `ticketdeveloperassignment` the id of the
developer to which a ticket is assigned is called `devid`,
 * which should be changed, while in the table `projectassignement`, the
column which needs to be changed is `pid`.
 *
 * To do this all in one shot, I'll write an code block (DO statement) to be
run database side.
 * This block will read a temporary table, which specifies the tables to
change and the columns to change for each table,
 * and change the specified columns with value `2` to `1` - transfering all
responsibilites from person with pid `2` to
 * the person with pid `1`.
 */
-- create a temporary table which specifies which tables to change,
-- and which columns within each table to change.
drop table if exists temp_tablestochange;
create temporary table temp_tablestochange(
    tbl varchar(50),
    coltochange varchar(50)
);
-- insert the tables/columns to change into this temporary table.
insert into temp_tablestochange values
 ('ticketdeveloperassignment', 'devid'),
 ('accountablebowner', 'pid'),
 ('projectassignment', 'pid');

-- Do statement - change the specified tables.
-- This works by formatting the string 'UPDATE <tbl> SET <col> = 1 where
<col> = 2' - and executing it as SQL.
DO $$DECLARE r record;
BEGIN
    FOR r IN SELECT tbl, coltochange from temp_tablestochange
    LOOP
        EXECUTE 'UPDATE ' ||r.tbl|| ' set ' ||r.coltochange|| ' = 1 where '
||r.coltochange|| ' = 2';
    END LOOP;
END$$;

-- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- -- --
--
/*
```

```
 * Overly ambitious developers/project managers may assign tickets to a
sprint
 * on the last day of the sprint's lifespan.
 *
 * It's not a reasonable thing to do, however, as it's not possible to
finish an entire
 * ticket in one day, and so the ticket should probably be kept saved until
the next sprint.
 *
 * These commands isolate only the tickets that were scheduled to a sprint
at least
 * one day before the sprint's closing date, and stores it in a temporary
table.
 * It then removes all the values in the `ticketsprintassignment` table, and
replaces them
 * with the values in the temporary table.
 *
 * This effectively cleans the `ticketsprintassignment` of values that leave
less than one
 * day to complete a ticket.
 */
-- create a temporary table of the ticket assignments that were assigned
-- to a sprint at least one day before the sprint's closing
create temporary table temp_reasonableticketschedules as (
    with assignmentinfo as (
        select tsa.tid,
            tsa.sprintnumber,
            tsa.teamname,
            tsa.datescheduled,
            s.startdate,
            s.enddate
        from ticketsprintassignment tsa
        inner join sprint s
            on tsa.sprintnumber = s.sprintnumber
            and tsa.teamname = s.teamname
        order by tid
    )
    select t.tid,
        t.sprintnumber,
        t.teamname,
        t.datescheduled
```

```
    from
    (
        select *,
            enddate - datescheduled as scheduledworktime
        from assignmentinfo
    ) t
     where scheduledworktime > 0
);
-- delete all values from the `ticketsprintassignment` table.
delete from ticketsprintassignment
    where true;
-- Replace these values with the values of the temporary table.
insert into ticketsprintassignment
    select * from temp_reasonableticketschedules;
```