COMP3311 21T1 Database Systems

Assignment 1

SQL, Views, PLpgSQL, Functions, Triggers

Last updated: **Saturday 27th Feb 08:33pm** (most recent updates are in [..])

Due : **Friday 19th March 17:00**

Aims

The aims of this assignment are to:

- formulate SQL queries;
- populate an RDBMS with a dataset, and analyse the data;
- design test data for testing SQL queries;
- create SQL views;
- understand the limitations of SQL queries; and
- create SQL functions, PLpgSQL and triggers (only when needed).

Description

This assignment is based on a simplified database to manage policy information for a motor vehicle insurance company. The company sells different types of tailor-made insurance policies with *coverages* that protect against loss, damage, injury etc. involving a motor vehicle (the *insured item*). For example, the company may sell greenslips, 3rd party property damage schemes or comprehensive schemes with tailor-made coverages.

An insurance policy is for one insured item with one or more coverages. In this assignment, we consider only the case where clients are private individuals (i.e. we do not consider corporate insurance schemes). A *policy* is sold directly by the company's staff (the agent). The *insured party* can be multiple people, such as a person and his/her spouse or a family.

The first step in selling insurance is to *draft* a policy, coordinated by the agent. A policy may have several coverages; for instance, fire, stolen, third-party liabilities, etc. The agent will liaise with the client to determine what coverages are required by the client, and enter this information to the database. After that, the insurance company has a rater team (that may contain more than one rater) who determines the rate that will be charged for each coverage. The rate is based on the vehicle, the particular coverage provided, and any qualifications (fine print) given by the insured party. If the rater team may refuse a coverage when no reasonable rate can be offered (e.g., the vehicle was too old to be covered, or very bad track record of the insured party).

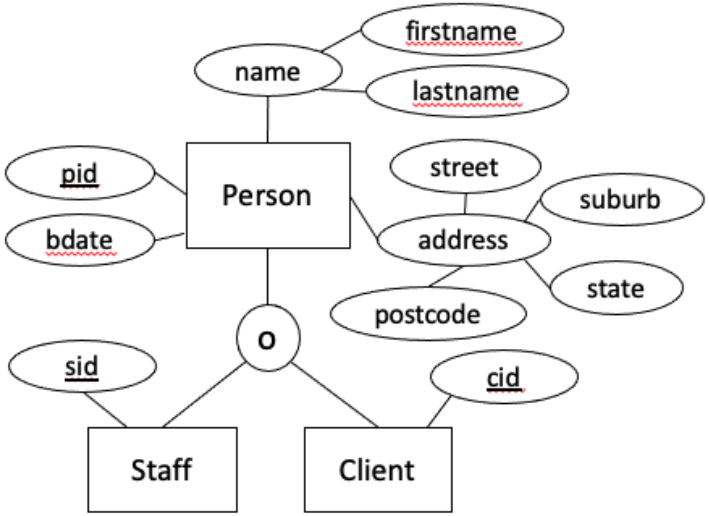
After all the coverages of a policy have been rated and approved by rater teams, the final step is to underwrite (approve) the policy by a team of underwriters (that may include more than one underwriters). Not all the policies will be approved without change. The underwriter team may choose to refuse some policies. In such cases, the agent will consult with the insured parties before making any modification. The agents, raters and underwriters are all staff members of the company. While a staff member can purchase a policy from the company, none of the insured parties of the policy can be the agent, a rater, or an underwriter of that policy.

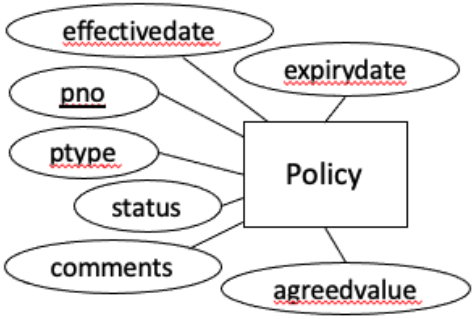
When a policy is refused by the underwriter team, the agent may discuss with the client and then submit updated coverages or client qualifications for a new round of rating and underwriting. Therefore, the procedures of an underwriting and rating may be processed several

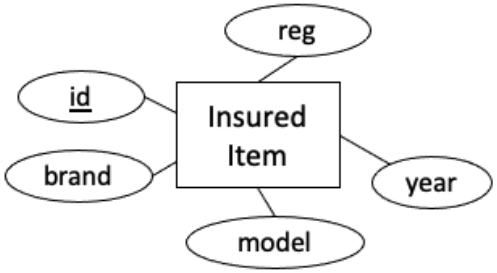
rounds (possibly on the same or different dates) by the same or different staff members. After a policy is underwritten, the client is assumed to pay and the policy will be enforced.

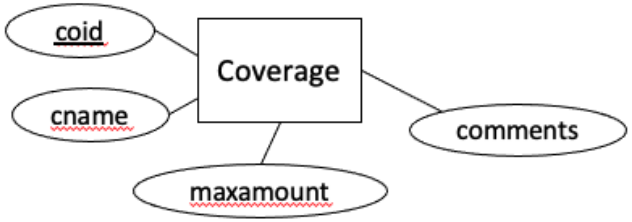
The above requirements plus additional constraints are captured and described by the ER schema below:

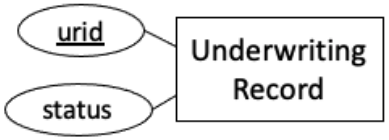
Entities:



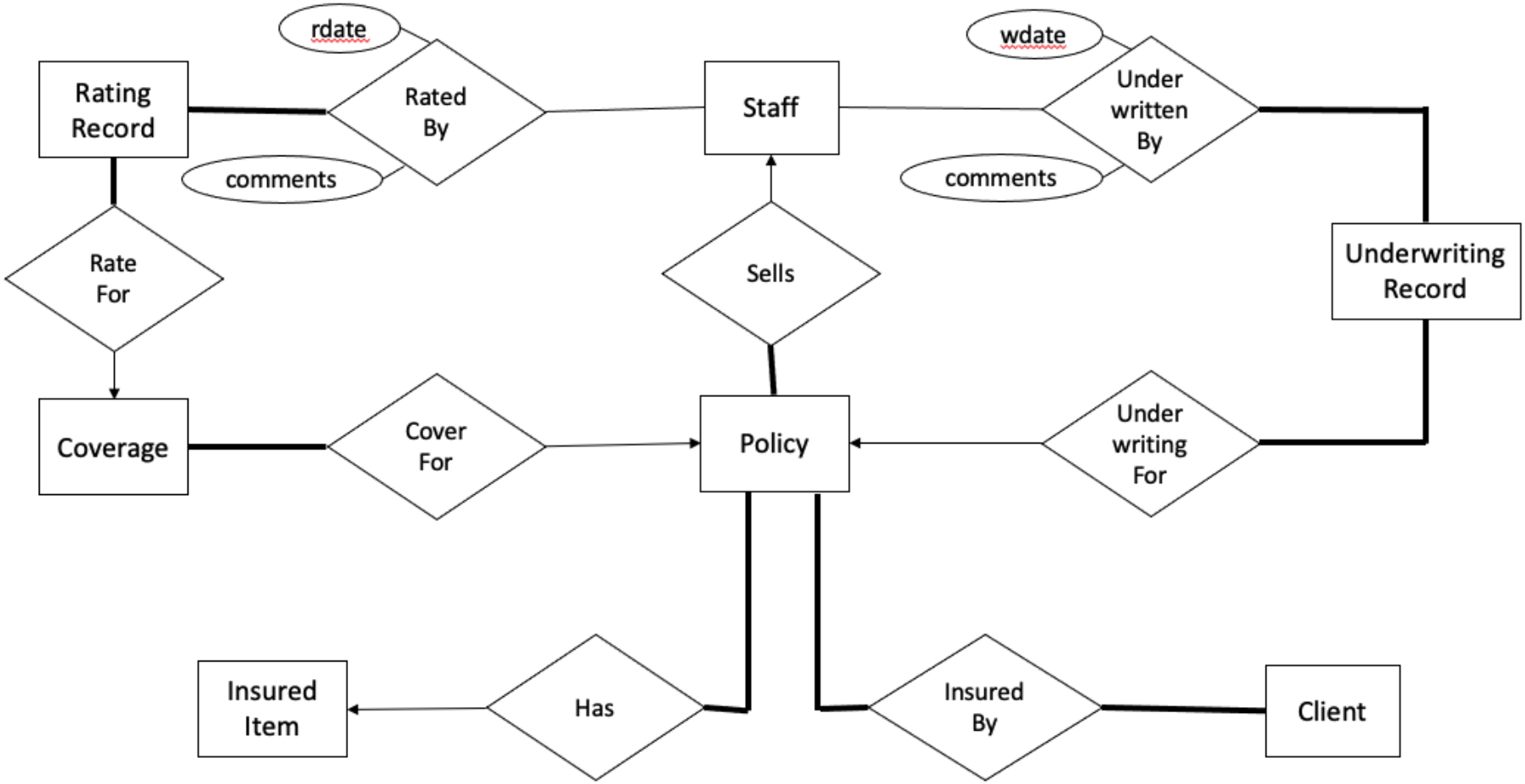








Relationships:



Further information:

1. *Person* stores personal details.
   - *Client* and *Staff* are specializations of *Person*.
   - *Person* may include people that are neither *Client* nor *Staff*.
2. *Policy* records policy details.
   - Here, **ptype** takes one of the 3 values - G, C, T, and P. These four values correspond to four different policy types - Greenslip, Comprehensive Insurance, Third Party Insurance, and Premium Options, where Premium Options is the policy that is totally customisable and can include any coverages.
   - The attributes **effectivedate** and **expirydate** specify the period during which a policy is active.
   - The attribute **agreedvalue** presents the amount of money that the company will pay to the client if the insured car has a total loss.
   - The attribute **status** takes one of the 6 values - D, RR, AR, RU, AU, E. These correspond to the 6 possible states of policy processing: once a policy is drafted its status is D. Then, the policy is sent to a rater team; and the status will be set to either RR or

AR according to a rating result; AR means an approval while RR means a refusal. Finally, the policy is sent to an underwriter team where AU means an approval and RU means a refusal. Once a policy is underwritten it will be enforced; consequently the status is set to E.

3. *Underwriting_record* presents the detail of each underwriting where **status** takes one of the possible three values, R, A, W, and O. These correspond to a refusal, an approval, a waiting (pending), and an obsolete/outdated approval. A previous approval for a policy will be updated to O when a new round of underwriting is taking place for that policy. Therefore, after several rounds of underwriting of a policy, there may be several refusals, waitings or obsolete approvals for the policy. However, there can only be at most one successful approval.
4. The attribute **status** in *Rating_record* is similar to that in *Underwriting_record*. The attribute **rate** in *Rating_record* tells a client the money that he/she has to pay to have the corresponding coverage.
5. *Coverage* stores the detail of a coverage. The attribute **cname** describes the coverage type (e.g., the third party insurance). The attribute **maxamount** specifies the maximal amount of money that the company will pay to a client under the coverage.
6. *Insured_item* stores the detail of insured vehicles.

The SQL definitions of the above schema, together with some sample data, are included in preload.sql. Based on this provided schema, you are required to answer the following questions by formulating SQL queries. You may create SQL functions or PLpgSQL to help you, if and only if the standard SQL query language is not expressive and powerful enough to satisfy a particular question. You may need to update the sample data or populate more data in order to test your queries. To enable auto-marking, your queries should be formulated as SQL views, using the view names and attribute names provided. If you wish, you may define and include additional, intermediate views if they make your solution simpler to derive / express. If order is specified, marks will only be granted if your solution output is in correct order and there are no duplicates in your output.

Queries

1. List all persons that are neither clients nor staff members. Order the result by pid in ascending order.

```
create or replace view Q1(pid, firstname, lastname) as ...
```

2. List all persons (including staff and clients) who have never been insured (wholly or jointly) by an enforced policy from the company. Order the result by pid in ascending order.

```
create or replace view Q2(pid, firstname, lastname) as ...
```

3. For each vehicle brand, list the vehicle insured by the most expensive policy (the premium, i.e., the sum of its approved coverages' rates). Include only the past and current enforced policies. Order the result by brand, and then by vehicle id, pno if there are ties, all in ascending order.

```
create or replace view Q3(brand, vid, pno, premium) as ...
```

4. List all the staff members who have not sell, rate or underwrite any policies that are/were eventually enforced. Note that policy.sid records the staff who sold the policy (i.e., the agent). Order the result by pid (i.e., Persion id) in ascending order.

```
create or replace view Q4(pid, firstname, lastname) as ...
```

5. For each suburb (by suburb name) in NSW, compute the number of enforced policies that have been sold to the policy holders living in the suburb (regardless of the policy effective and expiry dates). Order the result by Number of Policies (npolicies), then by suburb, in ascending order. Exclude suburbs with no sold policies. Furthermore, suburb names are output in all uppercase.

```
create or replace view Q5(suburb, npolicies) as ...
```

6. Find all past and current enforced policies which are rated, underwritten, and sold by the same staff member, and not involved any others at all. Order the result by pno in ascending order.

```
create or replace view Q6(pno, ptype, pid, firstname, lastname) as ...
```

7. The company would like to speed up the turnaround time of approving a policy and wants to find the enforced policy with the longest time between the first rater rating a coverage of the policy (regardless of the rating status), and the last underwriter approving the policy. Find such a policy (or policies if there is more than one policy with the same longest time) and output the details as specified below. Order the result by pno in ascending order.

```
create or replace view Q7(pno, ptype, effectivedate, expirydate, agreedvalue) as ...
```

8. List the staff members (their firstname, a space and then the lastname as one column called name) who have successfully sold policies (i.e., enforced policies) that only cover one brand of vehicle. Order the result by pid in ascending order.

```
create or replace view Q8(pid, name, brand) as ...
```

9. List clients (their firstname, a space and then the lastname as one column called name) who hold policies that cover all brands of vehicles recorded in the database. Ignore the policy status and include the past and current policies. Order the result by pid in ascending order.

```
create or replace view Q9(pid, name) as ...
```

10. Create a function that returns the total number of (distinct) staff that have worked (i.e., sells, rates, underwrites) on the given policy (ignore its status).

```
create or replace function staffcount(pno integer) returns integer ...
```

Return 0 if no policy exists for the given policy number.

11. Create a stored procedure that will start renewing an existing policy in the database.

```
create or replace procedure renew(pno integer) ...
```

The procedure will take in a policy number. If the policy is enforced and still effective, it will create a new policy in the Policy table with the same policy type, agreedvalue, comments but with the status set to D, the effectivedate set to the date of today, and the expirydate will be equal to the "today" + ("old expirydate" - "old effectivedata"). In other words, it will have the same effective duration as before. In addition, it will then update the old expirydate to today's date (to end the old policy). Finally, the same set of coverages will be created in the Coverage table for the newly created policy (with the same cname, maxamount and comments). These newly created coverages and policy will then be passed to the raters and underwriters for approval.

If the given policy is not currently effective (e.g., draft, refused, expired, etc.), create a new policy in the Policy table as above, but you do not set the old expirydate to today's date. Similarly, the same set of coverages will be created in the Coverage table for the newly created policy (with the same cname, maxamount and comments). These newly created coverages and policy will then be passed onto the raters and underwriters for approval.

Do nothing if no policy exists for the given policy number.

Also do nothing if there exists another currently effective (i.e., enforced and still effective as of today) policy of **the same policy type** for the same vehicle, for the given policy number.

12. A staff member can purchase an insurance policy from the company, but none of the insured parties of the policy can be the agent, a rater, or an underwriter of that policy. Create a trigger (or triggers) to enforce this constraint while allowing a staff member to purchase a policy.

Submission

Submission : Submit this assignment by doing the following:

Login to Course Web Site > Assignments > Assignment 1 > Assignment 1 Specification > Make Submission > upload `a1.sql` > [Submit]

You may also submit the assignment via the give command from CSE machines: `give cs3311 a1 a1.sql`

The `a1.sql` file should contain answers to all of the exercises for this assignment. It should be completely self-contained (without the schema and sample data) and able to load in a single pass.

Deadline : **Friday 19th March 17:00**

Late Penalty: Late submissions will have marks deducted from the maximum achievable mark at the rate of 2% of the total mark per hour that they are late (i.e., 48% per day).

Assessment

This assignment is worth a total of **16 marks**. Queries Q1-Q10 are each worth 1 mark. The stored procedure and the trigger questions are each worth 3 marks. The total will later be scaled to 20 percent for the course as described in the course outline.

Your submission (in a file called `a1.sql`) will be **auto-marked** on PostgreSQL in grieg.cse.unsw.edu.au to check:

- whether it is syntactically correct;
- if using SQL queries without creating a function or PLpgSQL unless it is necessary; and of course,
- if each query produces correct results.

What To Do Now

Make sure you read the above description thoroughly, and review and/or test out the provided schema and sample data preload.sql. The sample data is provided to help you quickly get started. While the same schema will be used to test your submission, a different dataset (that may be

larger, smaller, or totally different) may be used for auto-marking. Therefore, you may need to create your own or modify the provided data file to test your queries before submitting your assignment. Note that you DO NOT need to submit your data file as part of the submission..

Reminder: before you submit, ensure that your solution (`a1.sql`) will load into PostgreSQL without error if used as follows on `grieg`:

```
% dropdb a1
% createdb a1
% psql a1 -f preload.sql
% psql a1 -f a1.sql
... will produce notices, but should have no errors ...
% psql a1
... can start testing your solution ...
```

Important

Do not include any content from preload.sql, any schema definitions or data insertions in your a1.sql. If we have to fix errors in your solution before it will load, you will incur a 4 (out of total 16) mark "penalty". For example, if any of your view names or attribute names are different from the names specified above, you will incur a 4 mark "administrative penalty". Finally, it is entirely your responsibility to backup your solution. If you cannot submit the assignment because you lost data due to inadequate backup procedures, you will be treated as if you had not done the assignment in the first place.

Sample Output

Sample output from queries Q1 to Q10 based on the provided sample dataset in preload.sql is listed below. It is included in this specification so that you can verify the output formatting. Do not rely on them for correctness checking. We will not change the schema but may use a different / larger dataset in auto-marking. Therefore, before submitting your assignment solution, please test it thoroughly by consider all situations and exceptions carefully (e.g., by updating the data or creating more data).

```
a1=> select * from Q1;
 pid | firstname | lastname
-----+-----------+----------
  13 | Chris     | Evan
  14 | Tom       | Holland
  15 | Peter     | Pan
(3 rows)

a1=> select * from Q2;
 pid | firstname | lastname
-----+-----------+----------
   0 | Jack      | White
   1 | David     | Lee
   2 | Mary      | Jones
   3 | Vicky     | Donald
```

```
   4 | Vincent   | Thomas
   5 | Teresa    | Story
   6 | Alice     | Wang
   7 | David     | Bond
  11 | Nicole    | Kidman
  13 | Chris     | Evan
  14 | Tom       | Holland
  15 | Peter     | Pan
(12 rows)

a1=> select * from Q3;
 brand  | vid | pno | premium
--------+-----+-----+---------
 Honda  |   7 |   7 |     480
 Nissan |   0 |   2 |    1500
 Toyota |   4 |   5 |     400
(3 rows)

a1=> select * from Q4;
 pid | firstname | lastname
-----+-----------+----------
   2 | Mary      | Jones
   3 | Vicky     | Donald
   7 | David     | Bond
   8 | Frederick | Brown
   9 | Lucy      | Smiths
(5 rows)

a1=> select * from Q5;
    suburb    | npolicies
--------------+-----------
 ALEXENDRIA   |         1
 WOOLOOMOOLOO |         4
 NORTH SYDNEY |         5
(3 rows)

a1=> select * from Q6;
 pno | ptype | pid | firstname | lastname
-----+-------+-----+-----------+----------
   2 | C     |   0 | Jack      | White
(1 row)

a1=> select * from Q7;
```

```
 pno | ptype | effectivedate | expirydate | agreedvalue
-----+-------+---------------+------------+-------------
   6 | G     | 2018-12-16    | 2020-12-16 |       32500
(1 row)

a1=> select * from Q8;
 pid |     name     | brand
-----+--------------+--------
   0 | Jack White   | Nissan
   5 | Teresa Story | Honda
(2 rows)

a1=> select * from Q9;
 pid |      name
-----+-----------------
   8 | Frederick Brown
  12 | Rose Byrne
(2 rows)

a1=# select staffcount(4);
 staffcount
------------
          3
(1 row)
```