COMP-421 Project Phase 3

Group 12

March 2015

Andrew Fogarty 260535895 Lillio Mok 260456786 Zheng (Mason) Sun 260464047

-- This table keeps records of patients who were treated successfully with the illness, the day -- discharged, and the staff member treating the patient. CREATE TABLE OutPatientTreatment (patient_id INTEGER, ill_id char(8), treated by staff id INTEGER, ill until DATE NOT NULL, FOREIGN KEY (patient_id) REFERENCES Patient(patient id) ON DELETE CASCADE, FOREIGN KEY (ill id) REFERENCES Illness(ill_id), FOREIGN KEY (treated by staff id) REFERENCES Staff(staff_id) ON DELETE CASCADE, PRIMARY KEY (patient_id, ill_id, ill_until)); -- Procedure to log outpatient services for cured patient -- Find most recent staff member to treat patient; note that a patient who hasn't been treated and -- has a null ill_until date may not be deleted or considered to be an outpatient. CREATE OR REPLACE FUNCTION log_out_patient() RETURNS TRIGGER AS \$log_out_patient\$ **BFGIN** IF old.ill until IS NULL THEN RAISE EXCEPTION 'ill_until cannot be null'; END IF; **INSERT** INTO OutPatientTreatment(patient_id, ill_id, treated_by_staff_id, ill_until) SELECT old.patient id, old.ill id, staff id, old.ill until **FROM Treats** WHERE Treats.patient id = old.patient id ORDER BY Treats.since DESC LIMIT 1; RETURN old; END; \$log_out_patient\$ LANGUAGE 'plpgsql'; -- Response: CS421=> CREATE OR REPLACE FUNCTION log out patient() RETURNS TRIGGER AS \$log out patient\$ CS421\$> BEGIN CS421\$> IF old.ill_until IS NULL THEN CS421\$> RAISE EXCEPTION 'ill until

RAISE EXCEPTION 'ill until cannot be null';

CS421\$> END IF;

```
CS421$>
         INSERT
CS421$>
CS421$>
            INTO OutPatientTreatment(patient id, ill id, treated by staff id, ill until)
CS421$> SELECT old.patient_id, old.ill_id, staff_id, old.ill_until
CS421$>
          FROM Treats
          WHERE Treats.patient_id = old.patient_id
ORDER BY Treats.since DESC
CS421$>
CS421$>
CS421$>
          LIMIT 1;
CS421$>
CS421$>
           RETURN old;
CS421$> END;
CS421$> $log_out_patient$ LANGUAGE 'plpgsql';
CREATE TRICREATE FUNCTION
```

-- Trigger the procedure if we delete a patient

CREATE TRIGGER logOutPatient AFTER DELETE ON SufferingFrom FOR EACH ROW EXECUTE PROCEDURE log_out_patient();

-- Response

```
CS421=> CREATE TRIGGER logOutPatient AFTER DELETE ON SufferingFrom CS421-> FOR EACH ROW EXECUTE PROCEDURE log_out_patient(); CREATE TRIGGER
```

-- Examples; insert these if they are missing in the database

INSERT INTO SufferingFrom VALUES (691123, 'i0023325', '2014-05-15', '2014-06-01', 2000.00, 200.00, null);

INSERT INTO SufferingFrom VALUES (690000, 'i0023358', '2013-02-03', null, 60.00, 120.00, null);

-- Will not trigger because of null value

```
CS421=> DELETE FROM SufferingFrom WHERE patient_id = 690000 AND ill_id = 'i0023358'; ERROR: ill_until cannot be null
```

-- Will trigger

- -- This procedure logs high priority cases and recommends doctors to be assigned to each patient
- -- It first creates table HighPriorityPatient if it's not already been created
- -- It takes an argument type::VARCHAR(9) that may either be some version of 'urgency' or 'infectious'
- -- If urgency is chosen, patients of a high urgency will be considered to be high priority
- -- If infectious is chosen, patients with infectious diseases will be considered to be high priority
- -- It then polls a cursor created depending on the switches, and inserts the correct patients
- -- Both require that the patient has not been treated yet (i.e. ill_until date is NULL)
- -- It returns an integer of the number of inserted high priority cases

```
CREATE OR REPLACE FUNCTION log_high_priority_patients(type VARCHAR(9)) RETURNS INTEGER AS $log_high_priority_patients$

DECLARE
```

-- These are all the variables we need, with patient_id, ill_id, ill_since, urgency renamed

```
cur refcursor;

pat_id INTEGER;

il_id CHAR(8);

il_since DATE;

urge VARCHAR(32);

count INTEGER := 0;

BEGIN
```

-- Try to create the new relation, if it exists then move on.

```
BEGIN

CREATE TABLE HighPriorityPatient(

patient_id INTEGER,

ill_id char(8),

ill_since DATE,

recommended_staff_id INTEGER,

urgency VARCHAR(32),

contagious BOOLEAN,

FOREIGN KEY (patient_id)

REFERENCES Patient(patient_id) ON DELETE CASCADE,

FOREIGN KEY (ill_id)

REFERENCES Illness(ill_id),

FOREIGN KEY (recommended_staff_id)

REFERENCES Staff(staff_id),

PRIMARY KEY (patient_id, ill_id, ill_since)
);
```

EXCEPTION WHEN duplicate table THEN

END;

-- Case switches for the choice of high prioritisation based on urgency or infectiousness CASE

```
WHEN type like '%urgency%' OR type like '%Urgency%' THEN

OPEN cur FOR SELECT SufferingFrom.patient_id, SufferingFrom.ill_id, SufferingFrom.ill_since,
SufferingFrom.urgency FROM SufferingFrom WHERE (urgency LIKE '%Urgent%' OR urgency like
'%urgent%' AND ill until IS NULL);
```

```
WHEN type like '%infectious%' OR type like '%Infectious%' OR type like '%contagious%' OR type like
'%Contagious%' THEN
      OPEN cur FOR SELECT patient id, Illness.ill id, ill since, urgency FROM Illness INNER JOIN
SufferingFrom ON Illness.ill id = SufferingFrom.ill id WHERE contagious = TRUE AND ill until IS NULL;
    ELSE
      RAISE EXCEPTION 'Inapplicable high priority type.';
  END CASE;
  -- Look through the cursor
  LOOP
    FETCH cur INTO pat_id, il_id, il_since, urge;
    IF NOT FOUND THEN
      EXIT;
    END IF;
    -- Insert the new tuple, but ignore it if it already exists w.r.t. its primary key
      INSERT INTO HighPriorityPatient
      SELECT pat_id, il_id, il_since, NULL, urge, Illness.contagious
      FROM Illness
      WHERE Illness.ill id = il id;
      -- attempt to find a suitable doctor
      UPDATE HighPriorityPatient
      SET recommended_staff_id = SpecializesIn.staff_id
      FROM SpecializesIn
      WHERE SpecializesIn.ill_id = il_id
      AND HighPriorityPatient.patient_id = pat_id
      AND HighPriorityPatient.ill id = il id
      AND HighPriorityPatient.ill_since = il_since;
      count = count + 1;
    EXCEPTION WHEN unique violation THEN
    END;
  END LOOP;
  RETURN count;
END;
$log_high_priority_patients$ LANGUAGE 'plpgsql';
-- Applicable patients:
CS421=> SELECT * FROM SUFFERINGFROM where urgency like '%urgent%' or urgency like '%Urgent%' AND
ill until is NULL;
patient_id | ill_id | ill_since | ill_until | insurance_coverage | treatment_cost |
```

urgency

```
691126 | i0023313 | 2014-11-08 | | 691128 | i0023313 | 2014-11-23 | | 691129 | i0023366 | 2012-10-10 |
                                        2500.00 | 200.00 | Urgent
15000.00 | 200.00 | Urgent
5000.00 | 400.00 | very
    691129 | i0023378 | 2012-01-01 | 10000.00 | 500.00 | very
urgent I will die
(4 rows)
CS421=> SELECT * FROM SUFFERINGFROM INNER JOIN ILLNESS ON SUFFERINGFROM.ILL ID = ILLNESS.ILL ID
WHERE contagious = true AND ill until is NULL;
patient id | ill id | ill since | ill until | insurance coverage | treatment cost | urgency |
ill_id | ill_name | contagious | average_treatment_cost
690001 | i0023358 | 2013-02-03 |
                                    60.00 |
                                                            60.00 I
                                          1 t
200
1 t
                                                     200.00 | Urgent |
80.50
i0023313 | Gastroenteritis
    691126 | i0023313 | 2014-11-08 |
                                             2500.00 |
                                    | t
i0023313 | Gastroenteritis
   691125 | i0023313 | 2014-12-07 |
                                          80.50
200.00 | 80.50
i0023313 | Gastroenteritis
                                              t
2000.00 |
                                   691123 | i0023313 | 2015-03-07 |
                                            Ιt
i0023313 | Gastroenteritis
(7 rows)
```

-- To show that HighPriorityPatient does not exist

```
CS421=> SELECT * FROM HighPriorityPatient;
ERROR: relation "highprioritypatient" does not exist
LINE 1: SELECT * FROM HighPriorityPatient;
```

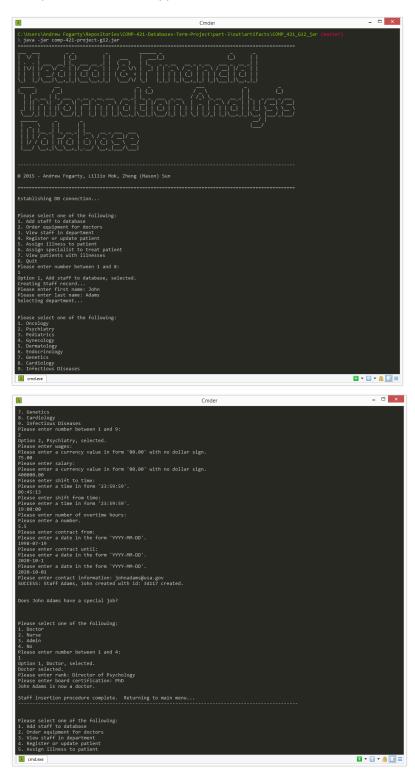
-- Responses after executing procedure

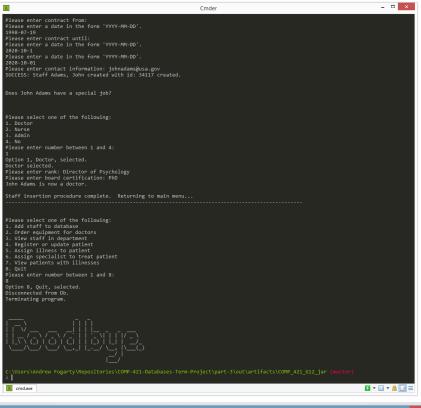
```
CS421=> SELECT log high priority patients('infectious');
NOTICE: CREATE TABLE / PRIMARY KEY will create implicit index "highprioritypatient pkey" for
table "highprioritypatient"
CONTEXT: SQL statement "CREATE TABLE HighPriorityPatient(
           patient id INTEGER,
           ill id char(8),
           ill since DATE,
           recommended staff id INTEGER,
           urgency VARCHAR(32),
           contagious BOOLEAN,
           FOREIGN KEY (patient id)
               REFERENCES Patient (patient id) ON DELETE CASCADE,
            FOREIGN KEY (ill id)
               REFERENCES Illness(ill_id),
            FOREIGN KEY (recommended staff id)
              REFERENCES Staff(staff id),
           PRIMARY KEY (patient id, ill id, ill since)
PL/pgSQL function "log high priority patients" line 12 at SQL statement
log_high_priority_patients
```

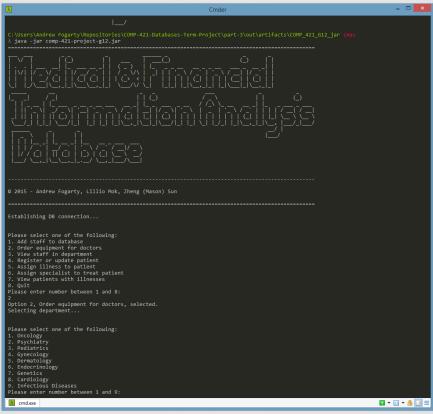
-- Note that repeated patients just imply that they have multiple untreated conditions requiring attention

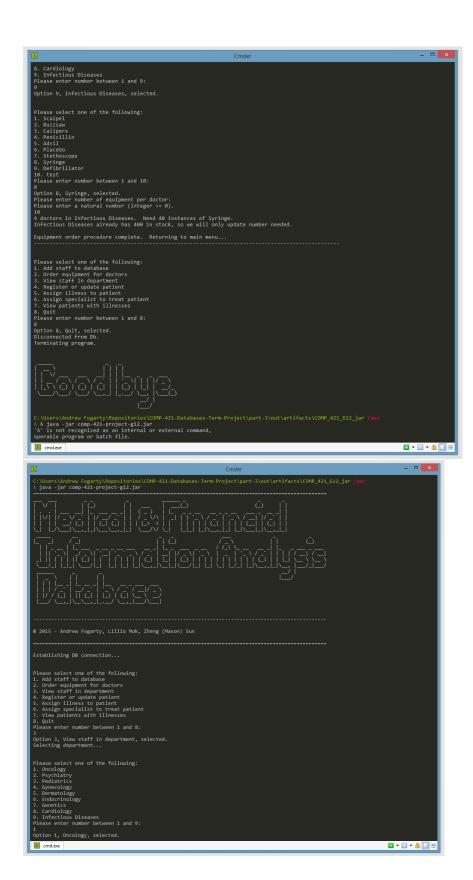
3)

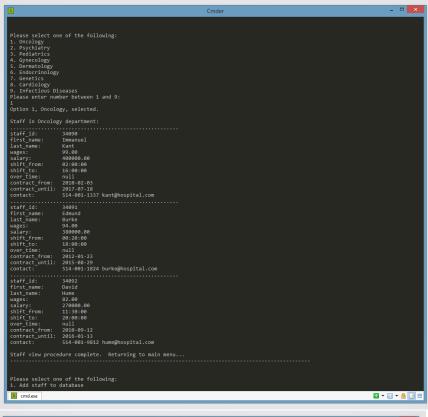
Please see attached for our application and executable. Due to ASCII art being produced on the command line, we cannot fit the output onto one pdf. This is also attached. Below are screenshots of the 7 procedures of our application.

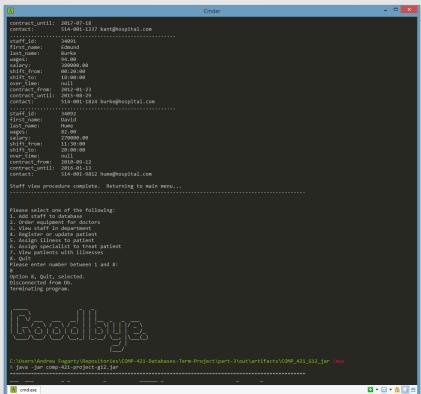


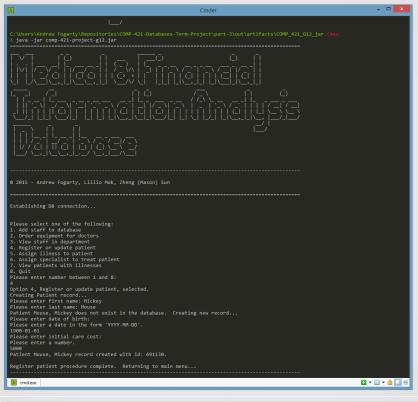




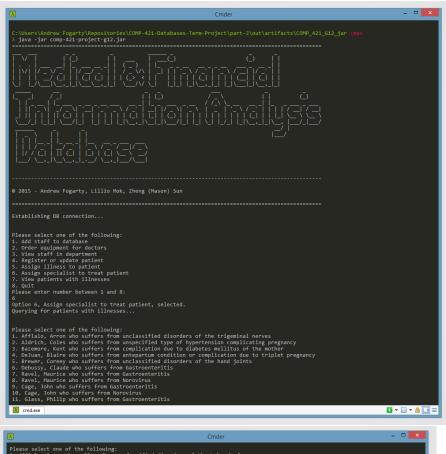


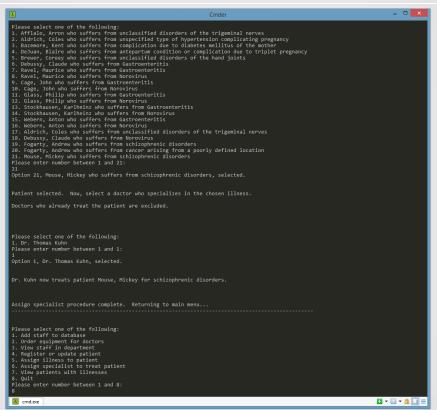




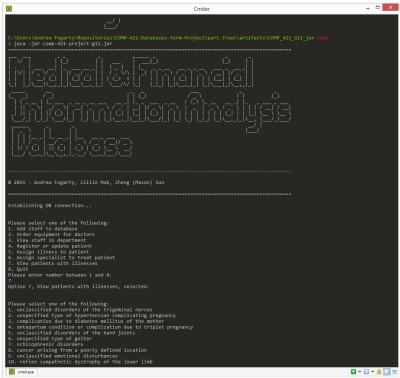


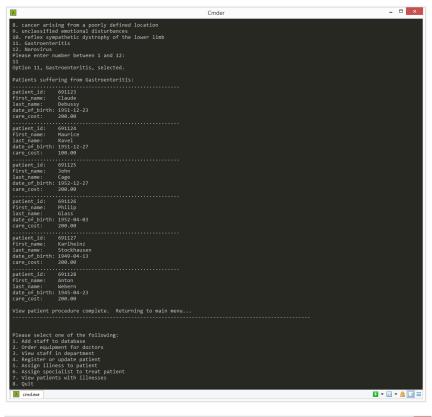


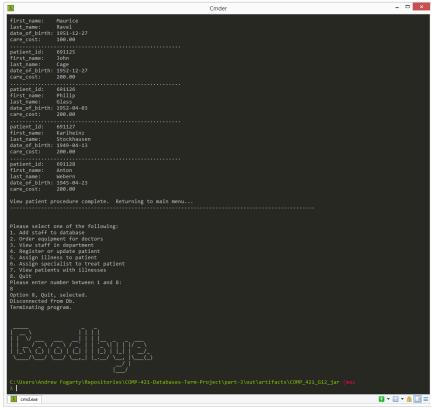












The complete scripts of these outputs are found under directory console-print-outs.

Please read README before running our executable.

Source code is under directory source.

4)

DROP INDEX toInd;

```
SELECT S.staff_id, first_name, last_name
FROM Staff S INNER JOIN Doctor D
ON S.staff id = D.staff id
  FULL OUTER JOIN Nurse N
  ON S.staff_id = N.staff_id
WHERE (
    shift_from <= '03:00:00' -- Shift ends between midnight and 3am
    OR shift_to <='03:00:00' -- Shift begins between midnight and 3am
    OR (shift_to > '03:00:00' AND shift_from > shift_to) -- Shift begins before midnight and ends after
3am
 );
CREATE INDEX fromInd ON Staff(shift from);
CREATE INDEX toInd ON Staff(shift to);
CS421=> CREATE INDEX fromInd ON Staff(shift from);
CREATE INDEX
CS421=> CREATE INDEX toInd ON Staff(shift to);
CREATE INDEX
-- These indices are useful as cost and amount needed as they require range searches,
-- and should shorten the time required for the queries. We don't think that it would make
-- a large amount of difference, however, because of limited records in the relation.
DROP INDEX fromInd;
```

- -- As we guessed, our results with the indexes were around 160ms, while the results without the
- -- indexes took around 130ms. This is, again, probably because we don't have the number of entries
- -- to justify the overhead for using indexes, and so there was little difference between the speeds.