Санкт-Петербургский Государственный Политехнический Университет

Институт Информационных Технологий и Управления

Кафедра Компьютерных Систем и Программных Технологий

**Отчет**

**о лабораторной работе №5**

**Дисциплина:** Базы данных

**Тема:** Триггеры

**Выполнил**: гр.43501/1 Саитов Илья

**Преподаватель:** Мяснов А.В.

Санкт-Петербург 2015

1. **Цель работы:**

Ознакомиться с возможностями реализации более сложной обработки данных на стороне сервера с помощью хранимых процедур и триггеров.

1. **Программа работы:**

* Создать два триггера: один триггер для автоматического заполнения ключевого поля, второй триггер для контроля целостности данных в подчиненной таблице при удалении/изменении записей в главной таблице.
* Создать триггер в соответствии с индивидуальным заданием, полученным у преподавателя.
* Создать триггер в соответствии с индивидуальным заданием, вызывающий хранимую процедуру.
* Продемонстрировать результаты преподавателю.

1. **Выполнение работы:**

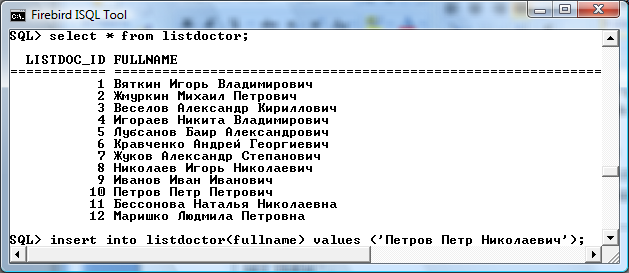
**а) Создадим два триггера: один триггер для автоматического заполнения ключевого поля, второй триггер для контроля целостности данных в подчиненной таблице при удалении/изменении записей в главной таблице:**

Создадим триггер, для автоматического заполнения ключевого поля ListDoc\_ID в таблице ListDoctor, если при добавлении новой записи, ListDoc\_ID не был указан (NULL). Создадим автоматический генератор ListDoc\_ID:

Листинг кода:

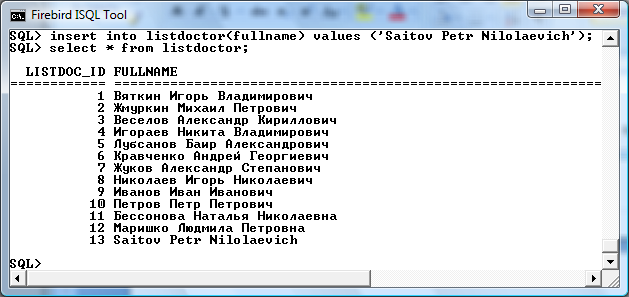
|  |
| --- |
| CREATE SEQUENCE LISTDOC\_GEN;  ALTER SEQUENCE LISTDOC\_GEN RESTART WITH 12;  SET TERM ^ ;  CREATE OR ALTER TRIGGER LISTDOC\_GEN FOR LISTDOCTOR  ACTIVE BEFORE INSERT POSITION 0  AS  BEGIN  IF (NEW.LISTDOC\_ID IS NULL) THEN  NEW.LISTDOC\_ID = GEN\_ID(LISTDOC\_GEN, 1);  END  ^  SET TERM ; ^ |

Испытаем написанный триггер в действии:



Выполним команду:

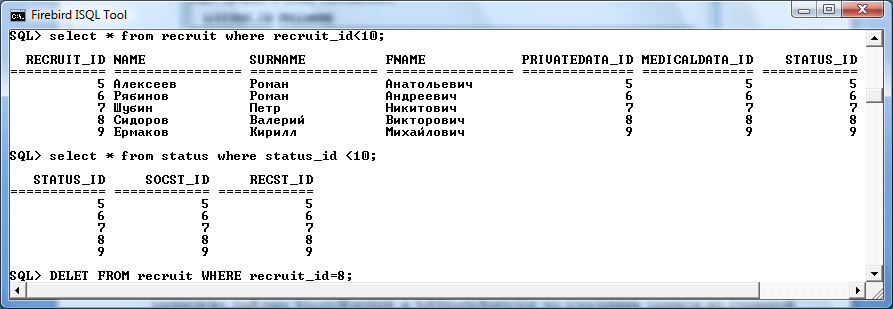
INSERT INTO listdoctor(fullname) VALUES ('Saitov Petr Nilolaevich');

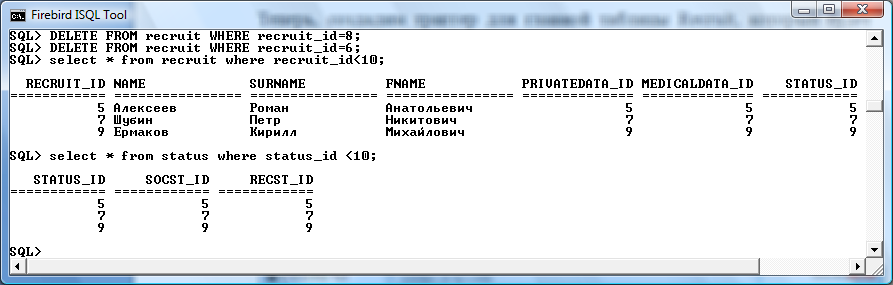


Теперь, создадим триггер для главной таблицы Recruit, который будет удалять все записи во всех таблицах при удалении человека из БД. Однако, для решения этой задачи понадобится 2 триггера: один, который будет удалять записи из таблиц StoryRecruit и MilitaryService до удаления записи из главной таблицы Recruit, так как они ссылаются внешним ключом на таблицу Recruit. Запись из таблицы Recruit нам удалить не удастся до тех пор, пока удаляемая запись в Recruit будет являться внешним ключом. Второй тригер будет удалять все остальные записи из таблиц MedicalData, Status, RecState, SocState, PrivateData, AddHome на которые ссылается удаляемая запись.

Листинг кода приведен в **приложении №1**.

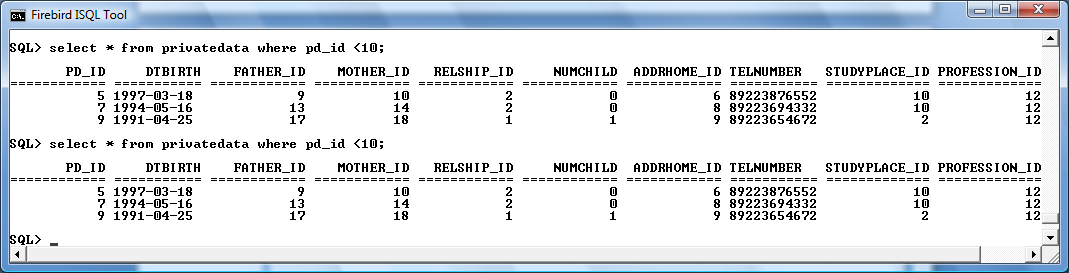
Для проверки работоспособности, попробуем удалить пару записей из таблицы Recruit командой DELETE:





DELETE FROM recruit WHERE recruit\_id=8;

DELETE FROM recruit WHERE recruit\_id=6;

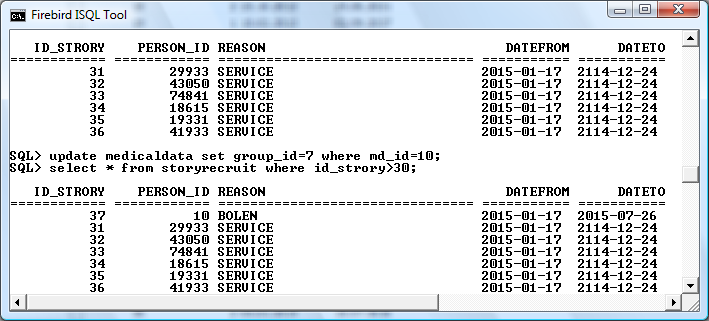


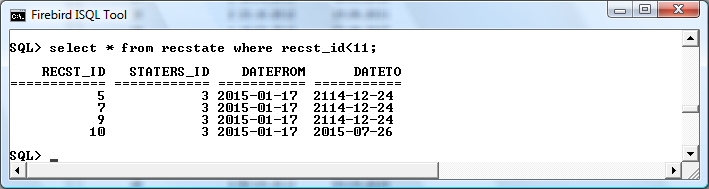
**б) При прохождении медицинксой комиссии после обследования всеми специалистами при наличии хотя бы одного заключения, не позволяющиего проходить воинскую службу будем создавать отсрочку от призыва в ВС на фиксированный период:**

В MedicalData при UPDATE или INSERT поля Group\_ID проверяется его значение. Если значение больше 6 (что соответствует группе В, Г, Д), то в таблице RECSTATE и STORYRECRUIT создается запись об отсрочке сроком на полгода.

Листинг:

|  |
| --- |
| SET TERM ^ ;  CREATE OR ALTER TRIGGER auto\_appeal FOR medicaldata  ACTIVE AFTER UPDATE POSITION 0  AS  declare variable TMP\_REC INT;  declare variable id INT;  declare variable TMP\_DATEFROM DATE;  declare variable TMP\_DATETO DATE;  BEGIN  SELECT current\_date FROM rdb$database  INTO :TMP\_DATEFROM;  TMP\_DATETO=TMP\_DATEFROM+190;  SELECT MAX(storyrecruit.id\_strory) FROM storyrecruit  INTO :id;  IF(new.group\_id>6) THEN  BEGIN  SELECT recruit.recruit\_id FROM recruit  WHERE (recruit.medicaldata\_id=old.md\_id)  INTO :TMP\_REC;  UPDATE recstate SET staters\_id=3 WHERE recst\_id=:TMP\_REC;  UPDATE recstate SET datefrom=:TMP\_DATEFROM WHERE recst\_id=:TMP\_REC;  UPDATE recstate SET dateto=:TMP\_DATETO WHERE recst\_id=:TMP\_REC;  id=id+1;  INSERT INTO storyrecruit(id\_strory, person\_id, reason, datefrom, dateto)  VALUES (:id, :TMP\_REC, 'BOLEN', :TMP\_DATEFROM, :TMP\_DATETO);  END  END ^  SET TERM ; ^ |

****

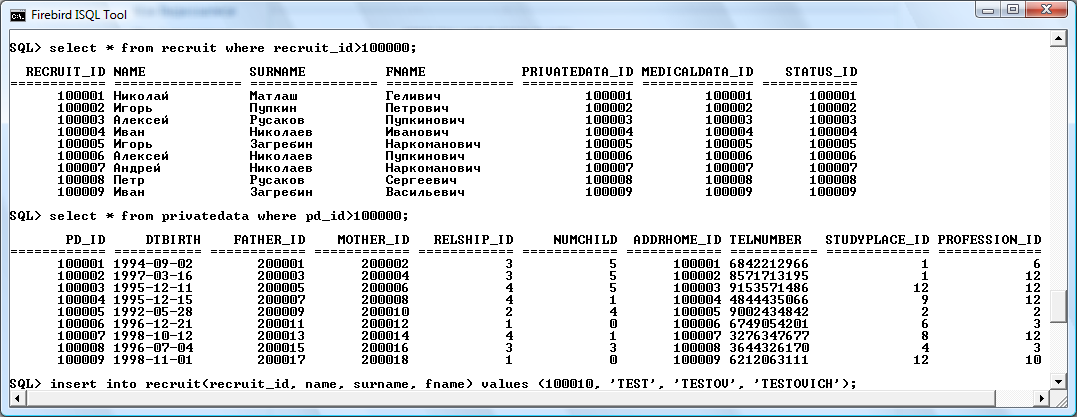
****

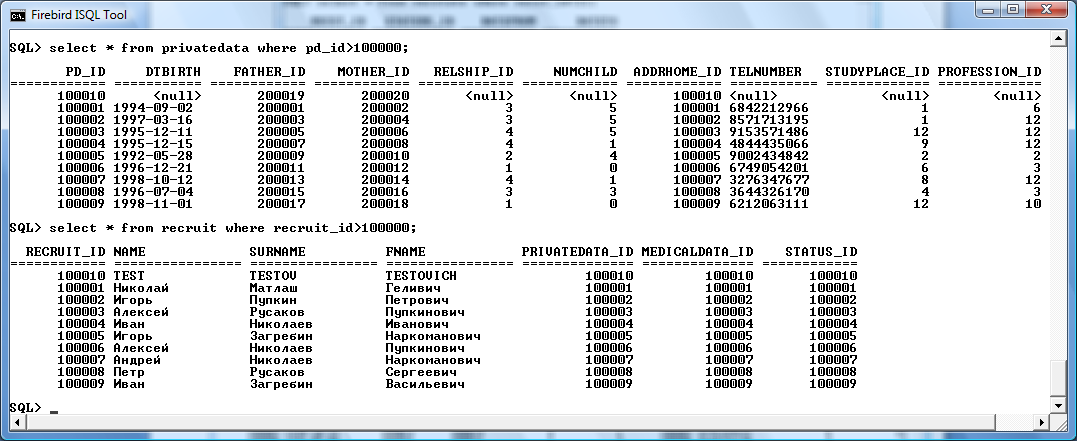
**в) При добавлении нового призывника будем создавать пустые записи для результатов медобследования и персональных данных:**

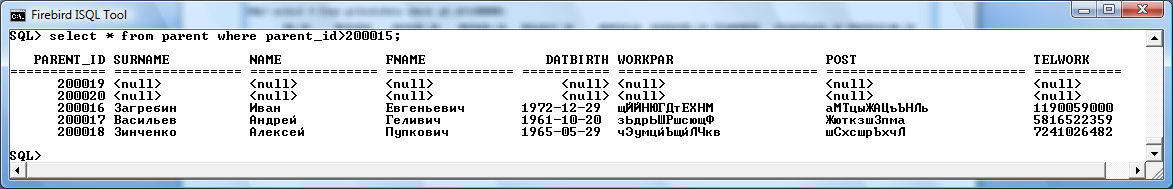
Для реализации данной задачи, нам потребуется 2 триггера – один работающий до выполнения команды INSERT, другой после. Автоматически, при добавлении человека в БД, создаются пустые записи в неключевых полях в таблицах ADDHOME, PARENT, MEDICALDATA, RECSTATE, SOCSTATE, STATUS. В ключевых полях автоматически создаются внешние ключи на созданные записи в таблицах.

Листинг кода SQL триггеров приведены в **приложении №2**.

Для эксперимента создадим запись в главной таблице Recruit:





****

Аналогично и в остальных таблицах. Как видно, все ключевые поля заполнены, остается лишь заполнить неключевые поля.

1. **Вывод:**

Выполнив лабораторную работу №5, мы ознакомились с возможностями реализации более сложной обработки данных на стороне сервера с помощью триггеров. Были созданы различные триггеры в соответствии с поставленным заданием, которые были успешно протестированы на работоспособность.

Одним из самых сложных заданиев было создание триггера для контроля целостности данных в подчиненной таблице при удалении/изменении записей в главной таблице. В роли главной таблицы выступала основная таблица Recruit, которая хранит ФИО и внешние ссылки на другие данные призывников. Трудность в том, что при удаление можно сделать не всегда, а только лишь при соблюдения условия того, что на данная запись не является внешним ключом. Большое количество таблиц и данных (100 000) в БД усложняло данную задачу объемом труда. Для решения данной задачи потребовалось 2 триггера, второй из них обеспечивал условие удаления записи.

Так или иначе, было выяснено, что триггера являются мощным инструментом при разработке БД, которые обеспечивают удобство и стабильность при работе БД в дальнейшем. Ведь это та же самая хранимая процедура, только которую пользователь не вызывает непосредственно. Ее исполнение обусловлено действием по модификации данных: добавлением INSERT, удалением DELETE строки в заданной таблице, или изменением UPDATE данных в определенном столбце заданной таблицы реляционной базы данных. Однако стоит помнить, злоупотребление данным инструментом ведет к падению производительности СУБД.

Цель данной работы была достигнута. Полученные знания при необходимости могут быть применены на практике.

**Приложение №1**. **Триггеры для удаления человека из БД:**

SET TERM ^ ;

CREATE OR ALTER TRIGGER del\_rec\_before FOR recruit

ACTIVE before DELETE POSITION 0

AS

BEGIN

DELETE FROM storyrecruit WHERE storyrecruit.person\_id = old.recruit\_id;

DELETE FROM militaryservice WHERE militaryservice.person\_id = old.recruit\_id;

END

^

SET TERM ; ^

SET TERM ^ ;

CREATE OR ALTER TRIGGER del\_rec FOR recruit

ACTIVE AFTER DELETE POSITION 0

AS

declare variable TMP\_DEL\_1 INT;

declare variable TMP\_DEL\_2 INT;

declare variable TMP\_DEL\_3 INT;

declare variable TMP\_DEL\_4 INT;

declare variable TMP\_DEL\_5 INT;

declare variable TMP\_DEL\_6 INT;

declare variable TMP\_DEL\_7 INT;

declare variable TMP\_DEL\_8 INT;

BEGIN

SELECT ther\_id FROM medicaldata WHERE medicaldata.md\_id = old.medicaldata\_id

INTO :TMP\_DEL\_1;

SELECT ent\_id FROM medicaldata WHERE medicaldata.md\_id = old.medicaldata\_id

INTO :TMP\_DEL\_2;

SELECT surg\_id FROM medicaldata WHERE medicaldata.md\_id = old.medicaldata\_id

INTO :TMP\_DEL\_3;

SELECT ocul\_id FROM medicaldata WHERE medicaldata.md\_id = old.medicaldata\_id

INTO :TMP\_DEL\_4;

SELECT derm\_id FROM medicaldata WHERE medicaldata.md\_id = old.medicaldata\_id

INTO :TMP\_DEL\_5;

SELECT neur\_id FROM medicaldata WHERE medicaldata.md\_id = old.medicaldata\_id

INTO :TMP\_DEL\_6;

SELECT psych\_id FROM medicaldata WHERE medicaldata.md\_id = old.medicaldata\_id

INTO :TMP\_DEL\_7;

SELECT conclusion\_id FROM medicaldata WHERE medicaldata.md\_id = old.medicaldata\_id

INTO :TMP\_DEL\_8;

DELETE FROM medicaldata WHERE medicaldata.md\_id = old.medicaldata\_id;

DELETE FROM doctor WHERE doctor.doc\_id=:TMP\_DEL\_1;

DELETE FROM doctor WHERE doctor.doc\_id=:TMP\_DEL\_2;

DELETE FROM doctor WHERE doctor.doc\_id=:TMP\_DEL\_3;

DELETE FROM doctor WHERE doctor.doc\_id=:TMP\_DEL\_4;

DELETE FROM doctor WHERE doctor.doc\_id=:TMP\_DEL\_5;

DELETE FROM doctor WHERE doctor.doc\_id=:TMP\_DEL\_6;

DELETE FROM doctor WHERE doctor.doc\_id=:TMP\_DEL\_7;

DELETE FROM doctor WHERE doctor.doc\_id=:TMP\_DEL\_8;

SELECT recst\_id FROM status WHERE status.status\_id = old.status\_id

INTO :TMP\_DEL\_1;

SELECT socst\_id FROM status WHERE status.status\_id = old.status\_id

INTO :TMP\_DEL\_2;

DELETE FROM status WHERE status.status\_id = old.status\_id;

DELETE FROM socstate WHERE socstate.socst\_id=:TMP\_DEL\_2;

DELETE FROM recstate WHERE recstate.recst\_id=:TMP\_DEL\_1;

SELECT mother\_id FROM privatedata WHERE privatedata.pd\_id = old.privatedata\_id

INTO :TMP\_DEL\_1;

SELECT father\_id FROM privatedata WHERE privatedata.pd\_id = old.privatedata\_id

INTO :TMP\_DEL\_2;

SELECT addrhome\_id FROM privatedata WHERE privatedata.pd\_id = old.privatedata\_id

INTO :TMP\_DEL\_3;

DELETE FROM privatedata WHERE privatedata.pd\_id = old.privatedata\_id;

DELETE FROM addrhome WHERE addrhome.ah\_id=:TMP\_DEL\_3;

DELETE FROM parent WHERE parent.parent\_id=:TMP\_DEL\_1;

DELETE FROM parent WHERE parent.parent\_id=:TMP\_DEL\_2;

END

^

SET TERM ; ^

**Приложение №2. Триггеры для создания пустых полей при добавлении человека в БД:**

SET TERM ^ ;

CREATE OR ALTER TRIGGER auto\_insert FOR recruit

ACTIVE BEFORE INSERT POSITION 0

AS

declare variable max\_private INT;

declare variable max\_parent INT;

declare variable max\_addr INT;

declare variable max\_med INT;

declare variable max\_doctor INT;

declare variable max\_status INT;

declare variable max\_socst INT;

declare variable max\_recst INT;

BEGIN

SELECT MAX(privatedata.pd\_id) FROM privatedata

INTO :max\_private;

SELECT MAX(parent.parent\_id) FROM parent

INTO :max\_parent;

SELECT MAX(addrhome.ah\_id) FROM addrhome

INTO :max\_addr;

SELECT MAX(medicaldata.md\_id) FROM medicaldata

INTO :max\_med;

SELECT MAX(doctor.doc\_id) FROM doctor

INTO :max\_doctor;

SELECT MAX(status.status\_id) FROM status

INTO :max\_status;

SELECT MAX(socstate.socst\_id) FROM socstate

INTO :max\_socst;

SELECT MAX(recstate.recst\_id) FROM recstate

INTO :max\_recst;

max\_addr = max\_addr+1;

INSERT INTO addrhome(ah\_id) VALUES (:max\_addr);

max\_parent = max\_parent+1;

INSERT INTO parent(parent\_id) VALUES (:max\_parent);

max\_private = max\_private+1;

INSERT INTO privatedata(pd\_id, father\_id, addrhome\_id) VALUES (:max\_private,:max\_parent, :max\_addr);

max\_parent = max\_parent+1;

INSERT INTO parent(parent\_id) VALUES (:max\_parent);

UPDATE privatedata SET mother\_id=:max\_parent WHERE pd\_id=:max\_private;

max\_socst = max\_socst+1;

INSERT INTO socstate(socst\_id) VALUES (:max\_socst);

max\_status = max\_status+1;

INSERT INTO status(status\_id,socst\_id) VALUES (:max\_status, :max\_status);

max\_recst = max\_recst+1;

INSERT INTO recstate(recst\_id) VALUES (:max\_recst);

UPDATE status SET recst\_id=:max\_recst WHERE status\_id=:max\_status;

max\_doctor = max\_doctor+1;

INSERT INTO doctor(doc\_id) VALUES (:max\_doctor);

max\_med = max\_med+1;

INSERT INTO medicaldata(md\_id, ther\_id) VALUES (:max\_med, :max\_doctor);

max\_doctor = max\_doctor+1;

INSERT INTO doctor(doc\_id) VALUES (:max\_doctor);

UPDATE medicaldata SET ther\_id=:max\_doctor WHERE md\_id =:max\_med;

max\_doctor = max\_doctor+1;

INSERT INTO doctor(doc\_id) VALUES (:max\_doctor);

UPDATE medicaldata SET ent\_id=:max\_doctor WHERE md\_id =:max\_med;

max\_doctor = max\_doctor+1;

INSERT INTO doctor(doc\_id) VALUES (:max\_doctor);

UPDATE medicaldata SET surg\_id=:max\_doctor WHERE md\_id =:max\_med;

max\_doctor = max\_doctor+1;

INSERT INTO doctor(doc\_id) VALUES (:max\_doctor);

UPDATE medicaldata SET ocul\_id=:max\_doctor WHERE md\_id =:max\_med;

max\_doctor = max\_doctor+1;

INSERT INTO doctor(doc\_id) VALUES (:max\_doctor);

UPDATE medicaldata SET derm\_id=:max\_doctor WHERE md\_id =:max\_med;

max\_doctor = max\_doctor+1;

INSERT INTO doctor(doc\_id) VALUES (:max\_doctor);

UPDATE medicaldata SET neur\_id=:max\_doctor WHERE md\_id =:max\_med;

max\_doctor = max\_doctor+1;

INSERT INTO doctor(doc\_id) VALUES (:max\_doctor);

UPDATE medicaldata SET psych\_id=:max\_doctor WHERE md\_id =:max\_med;

max\_doctor = max\_doctor+1;

INSERT INTO doctor(doc\_id) VALUES (:max\_doctor);

UPDATE medicaldata SET conclusion\_id=:max\_doctor WHERE md\_id =:max\_med;

END ^

SET TERM ; ^

SET TERM ^ ;

CREATE OR ALTER TRIGGER auto\_insert\_after FOR recruit

ACTIVE AFTER INSERT POSITION 0

AS

declare variable max\_private INT;

declare variable max\_med INT;

declare variable max\_status INT;

declare variable max\_rec INT;

BEGIN

SELECT MAX(privatedata.pd\_id) FROM privatedata

INTO :max\_private;

SELECT MAX(medicaldata.md\_id) FROM medicaldata

INTO :max\_med;

SELECT MAX(status.status\_id) FROM status

INTO :max\_status;

SELECT MAX(recruit.recruit\_id) FROM recruit

INTO :max\_rec;

UPDATE recruit SET privatedata\_id=:max\_private WHERE recruit\_id=:max\_rec;

UPDATE recruit SET medicaldata\_id=:max\_med WHERE recruit\_id=:max\_rec;

UPDATE recruit SET status\_id=:max\_status WHERE recruit\_id=:max\_rec;

END ^

SET TERM ; ^