Research Higher Degree Database

Physical Model

Prepared by:  
Thursdays **Group 4**

Sam Deane dean0109

Andrew Zschorn zsch0003

Version 0.1-DRAFT

17/5/2014

Created as part of the requirements for Advanced Database GE 2014

Contents

[1. Translate logical data model for target DBMS 1](#_Toc390295124)

[1.1. Select target DBMS 1](#_Toc390295125)

[1.2. Design base relations 1](#_Toc390295126)

[1.3. Design representation of derived data 1](#_Toc390295127)

[1.4. Design general constraints 2](#_Toc390295128)

[2. Design file organisations and indices 2](#_Toc390295129)

[2.1. Analyse transactions 2](#_Toc390295130)

[2.2. Select file organisations DONE 6](#_Toc390295131)

[2.3. Select indices REVIEW 7](#_Toc390295132)

[2.4. Estimate disk space requirements 556 NOT DONE 7](#_Toc390295133)

[3. Design user views 10](#_Toc390295134)

[4. Design security mechanisms 11](#_Toc390295135)

[5. Introduce controlled redundancy if necessary 11](#_Toc390295136)

[6. Create SQL scripts for data definition 12](#_Toc390295137)

[6.1. Create base tables, constraints and indexes 12](#_Toc390295138)

[6.2. Create views 23](#_Toc390295139)

[6.2.1. Utilities 23](#_Toc390295140)

[6.2.2. Views for all staff, including professional staff 24](#_Toc390295141)

[6.2.3. Views for academic staff 25](#_Toc390295142)

[6.2.4. Views for RHD administration staff 26](#_Toc390295143)

[6.3. Create procedure for setting user privileges 27](#_Toc390295144)

[6.4. Create triggers for tracking changes and enforcing constraints 28](#_Toc390295145)

[7. Create SQL scripts to populate all tables with data 36](#_Toc390295146)

[7.1. Populate Country relation 36](#_Toc390295147)

[7.2. Populate lookup relations 39](#_Toc390295148)

[7.3. Populate staff information 42](#_Toc390295149)

[7.4. Populate applicant and applicant information 44](#_Toc390295150)

[8. Create SQL scripts for required queries 50](#_Toc390295151)

[9. Monitor and tune the operational system 50](#_Toc390295152)

[10. Update test plan 50](#_Toc390295153)

[11. Create SQL scripts to test system 50](#_Toc390295154)

[12. Test operational system 68](#_Toc390295155)

[13. References 69](#_Toc390295156)

# Translate logical data model for target DBMS

## Select target DBMS

The target DBMS is MySQL, since this was known to be the target DBMS at the initialisation of the project, the previous two models that of the conceptual and logical have been designed to be compatible with MySQL, with limited amount of specific implementation required. This is seen (though noted not technically correct) in the form of adding variables applicable to MySQL (such as setting variables as VarChars in place of more general Strings variables) in the conceptual diagram.

The information gathered the in the previous three sections of requirements gathering and analysis, Conceptual model Diagram and documentation and logical model Diagram and documentation in their latest iteration have been reviewed and collected into a single information source.

The target, MySQL, DBMS has been studied revealing how to preform base transactions (such that Create, Read Update and Delete Base Relations are done for the most part through standard SQL (see <http://dev.mysql.com/doc/refman/5.0/en/differences-from-ansi.html>.)) and that most, if not all of the required functionality (that of. Keys, Domains and constraints) is available through the standard enterprise version as will be used in the final implementation of the database.

This was then used to produce the following **Relational database schema**

SCHEMA HERE MAYBE?

## Design base relations

**Implementing base relations**

The data base relation have been implemented using ISO SQL standard (Section 6.1) with some specific minor MySQL specific adjustments.

In implementating the base relations the following was adhered to

**Document design of base relations**

DBDL definitions of Relations

Changes

## Design representation of derived data

No derived data fields have been identified except those of the checklists of which will be adjust whenever a change is made through a background update of the DB by the application. This derived data exists to make referencing the completeness of an application quick and to provide hard coded information checklist as per the initial requirements.

It is presumed that other derived information, will be calculated as required. This include the age of an application, number of applications flagged, number of applications managed, application history etc.. To aid such queries, index have been placed on the relevant foreign and primary keys that are expected to be used often.

## Design general constraints

There are a few design constraints that have been implemented, these are

* Only those who can supervise can add themselves to the supervised by table. This is enforces through the use of permissions

General constraints such as text and dates being non empty is assumed to be enforced by the application interface. This is mainly because MySQL does not support constraints or partial indexes making enforcing of such constraints difficult. However there are workarounds using ‘Trigger’s but since these can have a significant impact on performance (running every time on an update/insert event if implemented in most cases) they have not be included unless deemed absolutely necessary.

# Design file organisations and indices

## Analyse transactions

In this section we analyse the most frequent transactions with respect to which base relations they operate on and which operations are performed. This will serve to guide us as to where we can expect bottlenecks in the running system, and therefore what will be the most appropriate design decisions to ensure good system performance.

The user transactions are shown in Table 1, Table 2 and Table 3. These transactions are cross-referenced to base relations in Table 4 and Table 5. Figure 1 shows the corresponding transaction pathways.

Table 1 User transactions for all staff, including professional

|  |  |
| --- | --- |
|  | Look up applicant + publications + degrees + visa Status + Associated documents by applicant name |
|  | Look up applicant’s applications by applicant name |
|  | Look up applicant’s applications by applicant email |
|  | Look up incomplete applications |
|  | Look up all correspondences relevant to an application |
|  | Create new applicant and associated application records |
|  | Look up which staff member updated an Application most recently |
|  | Check for any decision recorded about an application |
|  | Look up an existing application and attach a new standard type document to an application |
|  | Look up an existing application and attached a new exceptional type document to an application |
|  | Look up an existing application and list outstanding information (checklist). |
|  | Update the checklist to confirm that a mandatory information requirement has been met |
|  | Retrieve all on-going applications for which the user has made the most recent correspondence |
|  | Record making a decision about an application |
|  | Update the status of an application |

Table 2 User transactions for academic staff

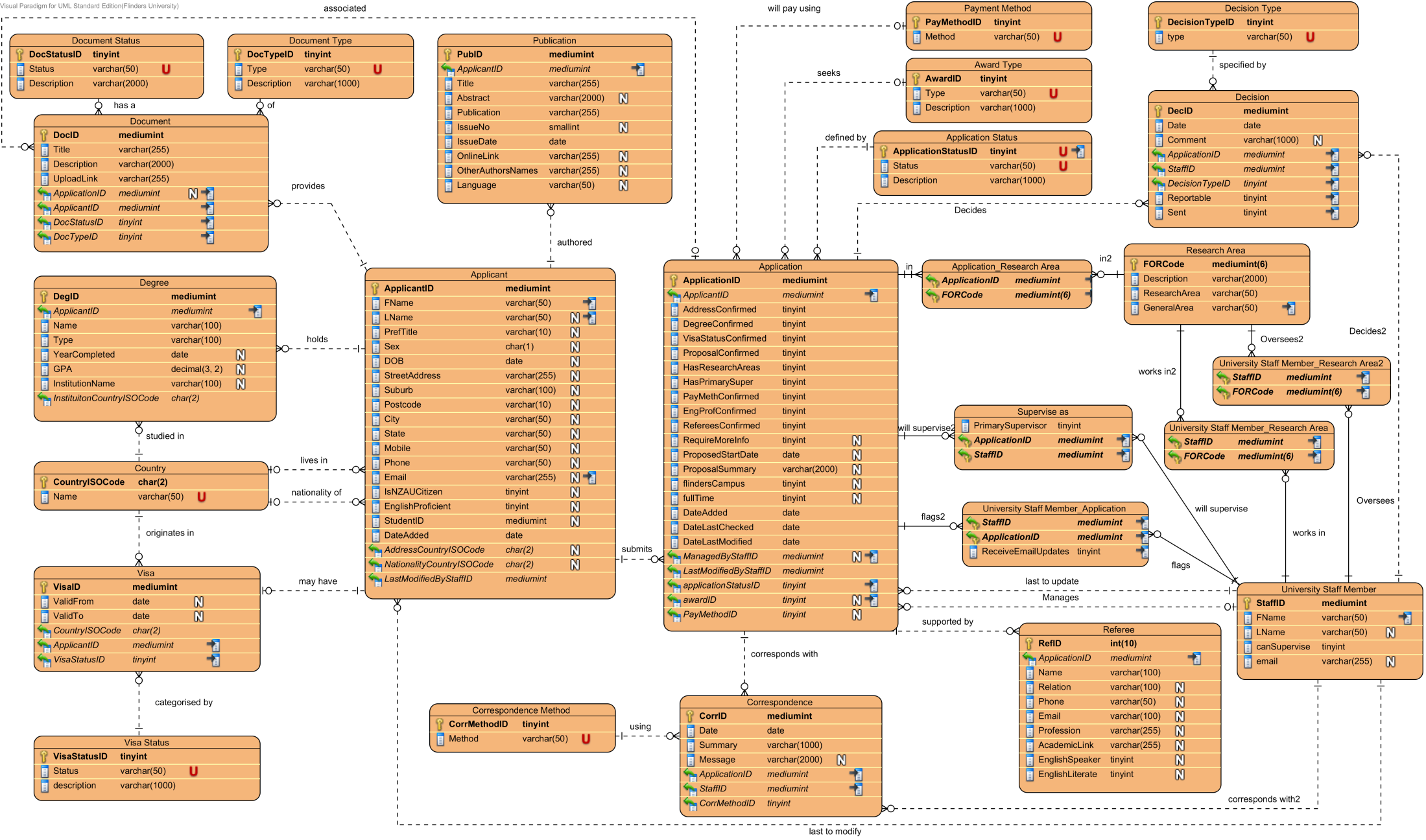
|  |  |
| --- | --- |
|  | Look up, add to, and delete from own current research areas |
|  | Search for all applications in certain research areas that have been added since a certain time |
|  | Flag interest in an application |

Table 3 User transactions for RHD co-ordination staff

|  |  |
| --- | --- |
|  | Retrieve all staff who have flagged an application, or have edited an application or applicant record most recently |
|  | Retrieve all ongoing applications |
|  | List all applications waiting for supervisor agreement |

Table 4 Cross referencing transactions and relations. Here shown are transactions for all staff, including professional staff





a1

a2

a3

a4

a5, i, j

b, c,

f, i

e, m

r

s

g, s

r

s

p

p

q

q

h, n

d, t, o

k, l, o

u

Figure 1 Transaction pathways

Table 5 Cross-referencing transactions and relations. Shown here are the transactions for academic and RHD co-ordination staff



Analysis of the database has revealed that nearly all transactions involve the Applicant and Applications relations, as these represent the core functionality of the database.

## Select file organisations DONE

The file organisations are grouped by storage engines in MySQL[[1]](#footnote-1). The default storage engine in MySQL 5.1 is called MyISAM. The MyISAM storage engine does not provide sufficient features to help ensure the integrity of a database. For instance, it cannot enforce foreign key constraints, nor is it transaction-safe, thus cannot be relied upon to support multiple concurrent users([Oracle Inc. 2014](#_ENREF_1" \o "Oracle Inc., 2014 #7)). InnoDB storage engine provides the required functionality for all relations. MySQL supports other storage engines, but they are all designed for specific cases that do not exist in the RHD database.

## Select indices REVIEW

<http://dev.mysql.com/doc/refman/5.7/en/optimization-indexes.html>

By Default MySQL places indices on the primary key (a clustered index for the InnoDB storage engine used here[[2]](#footnote-2)), these are also not null enabling fast queries.

Additional Indices have also been placed on the foreign keys of

* Primary Relations: application, applicant, university staff member and Research area

To enable fast joining between often joined relations

* Status and Type look-up Relations: Application Status, Document Status, Visa Status, Document Type and Decision Type

To assist in common transactions

Indexes have also been placed on the first and last names and emails of applicants and staff members as these will be the primary entry into the database relations, that is, all quires of the database are expected to start by searching for an applicants or staff members name or email.

To assist in the performance of such indices the primary key has been changed from int to medium int in non-lookup tables and unsigned tinyint in lookup tables that are expected to have less than 255 values. It is not expected that the database will have to hold more than 65,000 (unsigned small int) applicants, applications and their associated information but just in cases a medium int is used.

## Estimate disk space requirements 556 NOT DONE

Since all tables are set to use the INNODB storage engine, a clustered index is used as part of engine. This means that the records are physically stored (clustered) in a b-tree based on the index (left as the default primary key of each table). Each row (node of the b-tree) is then stored in a compact format (the default) reducing table space at the expense of some CPU overhead. It is also assumed that all characters are stored using the latin1 character Set with the latin1\_swedish\_ci. Collation (the MySQL INNODB engine defaults).

As such each row has

* 1 byte per TinyInt(lookup PKs), 2bytes per smallInt (postcode) 3bytes per mediumInt (main table PKs)
* ~5 bytes per index (the header), hard to gauge MySQL documentation is rather unspecific
* 4 bytes per decimal (GPA)
* CEILING(N/8) bytes for N null columns in the row
* L+1 bytes per L length of characters used in a varchar (as all varchars used are less than 255 so 1 byte to store the length and use the latin1 Set uses 1 byte per Character)
* 6+7 bytes for the transaction ID and roll pointer fields.
* 1 or 2 bytes per non null header (2 if “if part of the column is stored externally in overflow pages or the maximum length exceeds 255 bytes and the actual length exceeds 127 bytes”)

This enables the row size estimates to be calculated in the following way, i.e. for Applicant:

|  |  |
| --- | --- |
| **Date type** | **Estimated size in bytes** |
| varchars | 6\*(50+1)+(10+1)+2\*(255+1)+(100+1)+ |
| booleans | 3\*1+ |
| chars | 2\*1+ |
| smallint | 1\*2+ |
| mediumInt | 3\*3+ |
| date | 1\*3+ |
| secondary indexes | 3\*5 |
| header overheads | 6+7+2+5 |
| **TOTAL** | 984 |

|  |  |  |  |
| --- | --- | --- | --- |
| Table Name | Maximum possible size per row (bytes) | Expected max Size per row (bytes) (1/5max)\* | Expected average size per filled row (bytes) (~1/9max)\*\* |
| Applicant | 985 | 196 | 108 |
| Application | 2080 | 416 | 230 |
| Correspondence | 3055 | 610 | 338 |
| Decision | 1058 | 210 | 116 |
| Degree | 343 | 68 | 38 |
| Document | 2562 | 512 | 284 |
| Publication | 3113 | 622 | 344 |
| Referee | 896 | 178 | 98 |
| ResearchArea | 2131 | 426 | 236 |
| University Staff Member | 387 | 76 | 42 |
| Visa | 41 | 41 | 41 |
| Application\_  Research\_Area | 30 | 30 | 30 |
| Supervise As | 30 | 30 | 30 |
| University Staff Member \_Applicaiton | 31 | 31 | 31 |
| University Staff Member Research Area | 30 | 30 | 30 |
| University Staff Member Research Area 2 | 30 | 30 | 30 |
| Document Status | 2074 | 414 | 230 |
| Document Type | 2074 | 414 | 230 |
| Country | 73 | 50 | 40 |
| Visa Status | 32 | 32 | 28 |
| Correspondence Method | 72 | 50 | 40 |
| Payment Method | 72 | 50 | 40 |
| Award Type | 1074 | 214 | 118 |
| Application Status | 1074 | 214 | 118 |
| Decision Type | 72 | 50 | 40 |

\* and \*\* based on the assumption that names will use at most 10 characters with most names around 7 emails max around 45 and average around 20. With Join tables will be the max for each row.

To estimate the growth of the database the following assumptions have been made:

Applicants/Application based:

* It is expected that a maximum of 20 applications will be added per week
* 1 in 50 will be a repeat applicant
* Avg 1.75 degrees per applicant
* Avg 2 referees per application
* Avg 0.6 visas per applicant
* Avg 0.6 publication per applicant
* Avg 4 documents per applicant
* 4 research areas per application
* An application will be revised 2-10 times
* An application will involve 3-10 correspondences
* On average applications will be 70% complete (no proposal)
* On average applicnts will be 80% complete ()

Staff Based:

* A staff member will work in 5 areas
* A staff member will flag 5 applications per year
* Will state they will supervise 2 applications per year
* 50 staff members can supervise, 50 staff members can cannot supervise

Please note that the majority of these assumptions are purely speculative and should be considered in such a context

Hence the expected yearly growth for the relations is

|  |  |  |
| --- | --- | --- |
| Table Name | %complete | Increase per year (mb) |
| Applicant | 40 |  |
| Application |  |  |
| Correspondence |  |  |
| Decision |  |  |
| Degree |  |  |
| Document |  |  |
| Publication |  |  |
| Referee |  |  |
| ResearchArea |  |  |
| University Staff Member |  |  |
| Visa |  |  |
| Application\_  Research\_Area |  |  |
| Supervise As |  |  |
| University Staff Member \_Applicaiton |  |  |
| University Staff Member Research Area |  |  |
| University Staff Member Research Area 2 |  |  |

<http://dev.mysql.com/doc/refman/5.5/en/storage-requirements.html>

# Design user views

The database has four possible views each inheriting the previous view, as outlined by the initial requirements documentation.

These are

* Views for all staff, including professional, academic and RHD staff
  + Show all ongoing applications for which the current user staff member has an involvement. The definition of involvement includes playing a supervision role, having flagged the application, or being the staff member to most recently modify an application. The columns of this will be selected to allow the users to identify the application and applicant and the role they are playing (e.g. primary supervisor etc), and allow them to quickly determine what the next stage of developing the application.
* An ‘academic staff view’
  + A view for academic staff to list all recently added applicationsthat are in research areas that the current user has registered as working in.
* Views for RHD Co-ordination staff:
  + List all the ongoing applications and any staff member that has registered an involvement.
  + List all ongoing applications that currently haven’t had a primary supervisor assigned.

# Design security mechanisms

Deletion can only be performed by RHD Admin and this is assumed to occur very rarely.

Only staff members who can have the can supervise field set to true are able to state that they will supervise a RHD applicant for an application.

<http://www.greensql.com/content/mysql-security-best-practices-hardening-mysql-tips>

We have adopted a light-touch to security design in this RHD database.

As the RHD application process is not clearly designed, we want the database to support a diversity of workflows. This includes fostering volunteerism by allowing any staff member to take ownership of applications.

Hence, we grant all staff members have the privilege to SELECT, UPDATE and INSERT on all tables in the database, save for the following exceptions.

Only database administrators are granted the privilege to UPDATE and INSERT to the `University Staff Member` table.

There are areas of the database that ought not to be modified without the consent of certain others. For instance, no staff members should be registered as supervising an RHD application without their own consent. The other, similarly sensitive areas are: a staff member’s research areas and their oversight responsibilities for research areas; and which applications they have flagged.

To address this issue, we log all changes to the related tables so that the affected staff can be emailed summaries of such changes. This design has the benefit of allowing any user to make changes to these areas so potentially sharing widely the work of keeping the database up-to-date. In addition, those staff directly affected by such changes can be automatically forwarded notifications of those changes. By sharing more widely the responsibility for keeping this database up-to-date, we can minimise the extra time academic staff in particular need to spend on the system.

# Introduce controlled redundancy if necessary

One form of redundancy that we already have in our design is the checklist feature. This is the set of ten Boolean columns in the Application relation: AddressConfirmed, DegreeConfirmed, VisaStatusConfirmed, ProposalConfirmed, HasResearchAreas, HasPrimarySuper, PayMethodConfirmed, EngProfConfirmed, RefereesConfirmed and RequiresMoreInfo. All these values are derivable from other primary values in the schema. But we summarise them here to: allow the human users of the system to control theses values in a flexible way; and, less importantly, to increase the performance of the system so that these values do not need to be recalculated frequently.

We have decided to duplicate the Applicant(Email) attribute into the Application table. As email is the most common medium for communicating with applicants, it will almost always be present when an email is received with new information to update an application, and conversely when working on an application without being prompted by an email, the address should be retrieved by the system to help the user identify the applicant and send a new email to them. The email address can also be useful for users to search their email clients’ for messages sent and received about an application. Of course, care will have to be taken to ensure that this redundant information doesn’t become inconsistent. This is an example of duplicating a non-key attribute across a 1:\* relationship, which we expect to reduce the number of join operations on the Applicant, Application tables significantly. Also, email addresses aren’t likely to change very often, so keeping this information consistent across multiple redundant copies should not be too much work for a database system.

We also have considered other types of performance improvements by introducing controlled redundancy. We could combine one-to-one relationships, but we do not have any such relationships in our design. We could duplicate foreign key attributes across one-to-many relationships to reduce joins, but we do not have foreign key attributes used in joins sufficiently frequently to make this worthwhile. We could duplicate attributes in many-to-many relationships to reduce joins, but the many-to-many tables we have do not have many attributes. We considered introducing repeating groups, but decided there were no opportunities to do so in our design.

Creating extract tables is another way of increasing performance by introducing redundancy. We decided this would be best done in future once the system has been in use for a while. This would enable an accurate understanding of which frequently used reports slow the system down, and of those, for which it is appropriate to use potentially somewhat out-of-date data.

Another strategy to increase performance is partitioning relations. The most obvious application of this strategy to our design is to perform a horizontal partition on the Application table, separating all ongoing applications from completed applications. As almost all lookups on that table are expected to be on ongoing relations, rather than slow these down by scanning through one table that contains all applications ever created, it could search over a far smaller partition of only the ongoing relations.

Another way of achieving the same effect would be to create another copy of the Application table, and store all ongoing applications in the primary table, and move all complete applications to the secondary copy table. As this is not a common feature of database management systems.

# Create SQL scripts for data definition

## Create base tables, constraints and indexes

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

--

-- Create or re-create the RHD tables, constraints and indexes

--

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

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

-- Delete existing tables

SELECT "Dropping existing FK constraints" ;

ALTER TABLE Document DROP FOREIGN KEY associated;

ALTER TABLE Document DROP FOREIGN KEY provides;

ALTER TABLE Decision DROP FOREIGN KEY `specified by`;

ALTER TABLE Visa DROP FOREIGN KEY `originates in`;

ALTER TABLE Degree DROP FOREIGN KEY `studied in`;

ALTER TABLE Applicant DROP FOREIGN KEY `lives in`;

ALTER TABLE Applicant DROP FOREIGN KEY `nationality of`;

ALTER TABLE Application DROP FOREIGN KEY `will pay using`;

ALTER TABLE Application DROP FOREIGN KEY submits;

ALTER TABLE `Supervise as` DROP FOREIGN KEY `will supervise2`;

ALTER TABLE `Supervise as` DROP FOREIGN KEY `will supervise`;

ALTER TABLE `Application\_Research Area` DROP FOREIGN KEY `in`;

ALTER TABLE `Application\_Research Area` DROP FOREIGN KEY in2;

ALTER TABLE `University Staff Member\_Research Area` DROP FOREIGN KEY `works in`;

ALTER TABLE `University Staff Member\_Research Area` DROP FOREIGN KEY

`works in2`;

ALTER TABLE `University Staff Member\_Research Area2` DROP FOREIGN KEY Oversees;

ALTER TABLE `University Staff Member\_Research Area2` DROP FOREIGN KEY Oversees2;

ALTER TABLE Decision DROP FOREIGN KEY Decides;

ALTER TABLE Decision DROP FOREIGN KEY Decides2;

ALTER TABLE Visa DROP FOREIGN KEY `categorised by`;

ALTER TABLE Application DROP FOREIGN KEY `defined by`;

ALTER TABLE Document DROP FOREIGN KEY `has a`;

ALTER TABLE `University Staff Member\_Application` DROP FOREIGN KEY flags;

ALTER TABLE `University Staff Member\_Application` DROP FOREIGN KEY flags2;

ALTER TABLE Referee DROP FOREIGN KEY `supported by`;

ALTER TABLE Correspondence DROP FOREIGN KEY `corresponds with`;

ALTER TABLE Correspondence DROP FOREIGN KEY `corresponds with2`;

ALTER TABLE Document DROP FOREIGN KEY `of`;

ALTER TABLE Application DROP FOREIGN KEY Manages;

ALTER TABLE Degree DROP FOREIGN KEY holds;

ALTER TABLE Publication DROP FOREIGN KEY authored;

ALTER TABLE Visa DROP FOREIGN KEY `may have`;

ALTER TABLE Application DROP FOREIGN KEY seeks;

ALTER TABLE Application DROP FOREIGN KEY `last to update`;

ALTER TABLE Applicant DROP FOREIGN KEY `last to modify`;

ALTER TABLE Correspondence DROP FOREIGN KEY `using`;

SELECT "Dropping tables" ;

DROP TABLE IF EXISTS `Decision Type`;

DROP TABLE IF EXISTS `Payment Method`;

DROP TABLE IF EXISTS Application;

DROP TABLE IF EXISTS `Document Status`;

DROP TABLE IF EXISTS Publication;

DROP TABLE IF EXISTS Degree;

DROP TABLE IF EXISTS Referee;

DROP TABLE IF EXISTS Document;

DROP TABLE IF EXISTS Applicant;

DROP TABLE IF EXISTS Visa;

DROP TABLE IF EXISTS `Visa Status`;

DROP TABLE IF EXISTS Correspondence;

DROP TABLE IF EXISTS Decision;

DROP TABLE IF EXISTS `Research Area`;

DROP TABLE IF EXISTS `University Staff Member`;

DROP TABLE IF EXISTS `Correspondence Method`;

DROP TABLE IF EXISTS `Application Status`;

DROP TABLE IF EXISTS `Document Type`;

DROP TABLE IF EXISTS `Award Type`;

DROP TABLE IF EXISTS Country;

DROP TABLE IF EXISTS `Supervise as`;

DROP TABLE IF EXISTS `Application\_Research Area`;

DROP TABLE IF EXISTS `University Staff Member\_Research Area`;

DROP TABLE IF EXISTS `University Staff Member\_Research Area2`;

DROP TABLE IF EXISTS `University Staff Member\_Application`;

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

-- Create tables

SELECT "Creating tables" ;

CREATE TABLE `Decision Type` (

DecisionTypeID mediumint NOT NULL AUTO\_INCREMENT,

type varchar(50) NOT NULL UNIQUE comment 'the type of decision/comment made',

PRIMARY KEY (DecisionTypeID)

) comment='the possible types of decisions/comments that can be made'

ENGINE=InnoDB;

CREATE TABLE `Payment Method` (

PayMethodID mediumint NOT NULL AUTO\_INCREMENT,

Method varchar(50) NOT NULL UNIQUE comment 'Method of payment, e.g. scholarship, letter of financial support etc.',

PRIMARY KEY (PayMethodID)

)

comment='the possible payment methods of a Research higher degree'

ENGINE=InnoDB;

CREATE TABLE Application (

ApplicationID mediumint NOT NULL AUTO\_INCREMENT,

ApplicantEmail varchar (100)

comment 'Denormalised column to improve retrieval of applications via email address',

ApplicantID mediumint NOT NULL

comment 'the ID of the applicant who proposed this application',

AddressConfirmed tinyint(1) NOT NULL comment 'All contact details appear valid',

DegreeConfirmed tinyint(1) NOT NULL

comment 'The degree is a recognised degree of the institution',

VisaStatusConfirmed tinyint(1) NOT NULL

comment 'The visa status is backed by an official document',

ProposalConfirmed tinyint(1) NOT NULL

comment 'The proposal is contains appropriate detail',

HasResearchAreas tinyint(1) NOT NULL

comment 'Has nominated research areas relevant to the proposal',

HasPrimarySuper tinyint(1) NOT NULL

comment 'Has the required number of supervisors',

PayMethConfirmed tinyint(1) NOT NULL

comment 'The payment method is backed by an official document',

EngProfConfirmed tinyint(1) NOT NULL

comment 'The applicant has some level of English literacy',

RefereesConfirmed tinyint(1) NOT NULL

comment 'The referees details appear to be correct',

RequireMoreInfo tinyint(1),

ProposedStartDate date

comment 'The date the applicant prefers to start the RHD (Entered as 1/1/## for S1 and 1/7/## for S2)',

ProposalSummary varchar(2000)

comment 'What the proposal is about',

flindersCampus tinyint(1)

comment 'the applicant wants to complete the degree a main campus',

fullTime tinyint(1) comment 'the applicant wants to undergo the degree full time',

DateAdded date NOT NULL comment 'the date the application was added',

DateLastChecked date NOT NULL

comment 'the date the application was last checked',

DateLastModified date NOT NULL

comment 'the date the application was last modified',

ManagedByStaffID mediumint

comment 'The staff ID of the staff member who has been personally assigned to manage this application (since it may not be used it is nullable)',

LastModifiedByStaffID mediumint NOT NULL

comment 'the staff member ID of the last person to modify the application (all modifications are recorded in the decision table)',

applicationStatusID mediumint NOT NULL,

awardID mediumint,

PayMethodID mediumint,

PRIMARY KEY (ApplicationID),

INDEX (ApplicantID),

INDEX (applicationStatusID),

INDEX (awardID),

INDEX (ApplicantEmail)

)

comment='Holds the application details including a checklist of recorded information and applicaiton stage'

ENGINE=InnoDB;

CREATE TABLE `Document Status` (

DocStatusID mediumint NOT NULL AUTO\_INCREMENT,

Status varchar(50) NOT NULL UNIQUE comment 'Official and translation status of a document associated to an Applicant',

Description varchar(2000) NOT NULL comment 'the details and implication of this status',

PRIMARY KEY (DocStatusID)

)

comment='the possible statuses of a document'

ENGINE=InnoDB;

CREATE TABLE Publication (

PubID mediumint NOT NULL AUTO\_INCREMENT comment 'The primary key that uniquely identifies the publication',

ApplicantID mediumint NOT NULL comment 'the ID of the applicant who authored the publication',

Title varchar(255) NOT NULL comment 'The title of the publication',

Abstract varchar(2000) comment 'A abstract/description of the publication',

Publication varchar(255) NOT NULL comment 'The journal/magazine publisher ',

IssueNo mediumint comment 'The issue/edition number of the publication',

IssueDate date NOT NULL comment 'The date the publication was issued',

OnlineLink varchar(255) comment 'An online link to the publication',

OtherAuthorsNames varchar(255) comment 'Other authors of the publication',

Language varchar(50) comment 'language of the publication',

PRIMARY KEY (PubID))

ENGINE=InnoDB;

CREATE TABLE Degree (

DegID mediumint NOT NULL AUTO\_INCREMENT comment 'The primary key that uniquely identifies the degree',

ApplicantID mediumint NOT NULL comment 'the ID of the applicant who holds this degree',

Name varchar(100) NOT NULL comment 'The title of the degree',

Type varchar(100) NOT NULL comment 'The type of the degree -Â Â Could add specific types',

YearCompleted date comment 'The year the degree was completed or will be completed',

GPA mediumint comment 'The GPA of the degree',

InstitutionName varchar(100) comment 'The name of the institution',

InstitutionCountryISOCode char(2) NOT NULL comment 'the country the institution is based in',

PRIMARY KEY (DegID),

INDEX (ApplicantID)

)

comment='Any Degrees already held by the applicant'

ENGINE=InnoDB;

CREATE TABLE Referee (

RefID mediumint NOT NULL AUTO\_INCREMENT comment 'The primary key that uniquely identifies the referee supporting an application',

ApplicationID mediumint NOT NULL,

Name varchar(100) NOT NULL comment 'The full name of the referee',

Relation varchar(100) comment 'The referees relation to the applicant',

Phone varchar(50) comment 'The referees phone number',

Email varchar(100) comment 'The referees email address',

Profession varchar(255) comment 'The referees profession',

AcademicLink varchar(255) comment 'The referees professional page (linked in or university)',

EnglishSpeaker tinyint(1) comment 'If the Referee can speak English',

EnglishLiterate tinyint(1) comment 'If the Referee can read and write in English',

PRIMARY KEY (RefID)

)

comment='A referee for a application'

ENGINE=InnoDB;

CREATE TABLE Document (

DocID mediumint NOT NULL AUTO\_INCREMENT comment 'The primary key that uniquely identifies the document',

Title varchar(254) NOT NULL comment 'The title of the document',

Description varchar(2000) NOT NULL comment 'A specific summary related to the document i.e. valid till 2015 etc.',

UploadLink varchar(254) NOT NULL comment 'An link to a version uploaded and stored on the university servers',

ApplicationID mediumint,

ApplicantID mediumint NOT NULL comment 'the ID of the applicant who provided this document',

DocStatusID mediumint NOT NULL,

DocTypeID mediumint NOT NULL,

PRIMARY KEY (DocID),

INDEX (ApplicationID),

INDEX (ApplicantID),

INDEX (DocStatusID),

INDEX (DocTypeID)

)

comment='Links to any relevant documents along with descriptions,

types and statuses.'

ENGINE=InnoDB;

CREATE TABLE Applicant (

ApplicantID mediumint NOT NULL AUTO\_INCREMENT

comment 'The primary key that uniquely identifies the applicant',

FName varchar(50) NOT NULL comment 'First name',

LName varchar(50) comment 'Last name',

PrefTitle varchar(10) comment 'Title Mr,

Mrs,

Miss,

Dr. \*',

Sex tinyint(1) comment 'The sex of the applicant',

DOB date comment 'Date of birth',

StreetAddress varchar(255) comment 'Residence number and street of residence',

Suburb varchar(100) comment 'The suburb of residence',

Postcode mediumint comment 'The postcode of residence',

City varchar(50) comment 'The city or town of residence',

State varchar(50) comment 'The State of residence',

Mobile varchar(50) comment 'Mobile phone number',

Phone varchar(50) comment 'Landline phone number',

Email varchar(100) comment 'The email address of the applicant',

IsNZAUCitizen tinyint(1) comment 'Is a new Zealand or Australian citizen, a check to see if visa information is required \*\*\*',

EnglishProficient tinyint(1) comment 'English ability',

StudentID mediumint comment 'The flinders university student id if they are or have been enrolled at flinders university',

DateAdded date NOT NULL comment 'The date the applicant was added to the system',

AddressCountryISOCode char(2),

NationalityCountryISOCode char(2),

LastModifiedByStaffID mediumint NOT NULL comment 'the staff member ID of the last person to modify the applicant (all modifications are recorded in the decision table)',

PRIMARY KEY (ApplicantID),

INDEX (FName),

INDEX (LName),

INDEX (Email)

)

comment='Holds the applicant specific details'

ENGINE=InnoDB;

CREATE TABLE Visa (

VisaID mediumint NOT NULL AUTO\_INCREMENT comment 'The primary key that uniquely identifies the visa',

ValidFrom date comment 'When the visa is valid from',

ValidTo date comment 'When the visa is valid to',

CountryISOCode char(2) NOT NULL comment 'the applicant country, the visa is granted to',

ApplicantID mediumint NOT NULL comment 'the ID of the applicant who holds or may hold this visa',

VisaStatusID mediumint NOT NULL,

PRIMARY KEY (VisaID),

INDEX (ApplicantID),

INDEX (VisaStatusID)

)

comment='The applicants visa details'

ENGINE=InnoDB;

CREATE TABLE `Visa Status` (

VisaStatusID mediumint NOT NULL AUTO\_INCREMENT,

Status varchar(50) NOT NULL UNIQUE comment 'the status of the visa application',

description varchar(1000) NOT NULL comment 'a description of the status of the visa',

PRIMARY KEY (VisaStatusID)

)

comment='the possible statuses of the visa application'

ENGINE=InnoDB;

CREATE TABLE Correspondence (

CorrID mediumint NOT NULL AUTO\_INCREMENT,

`Date` date NOT NULL comment 'The date the correspondence was made/received',

Summary varchar(1000) NOT NULL comment 'A small summary of the Correspondence',

Message varchar(2000) comment 'The actual message contained in the correspondence',

ApplicationID mediumint NOT NULL comment 'the application ID the correspondence is in relation to',

StaffID mediumint NOT NULL comment 'the staff ID of the staff member the correspondence is to/from',

CorrMethodID mediumint NOT NULL,

PRIMARY KEY (CorrID),

INDEX (ApplicationID),

INDEX (StaffID)

)

comment='Correspondence between the Applicant and University Staff Member'

ENGINE=InnoDB;

CREATE TABLE Decision (

DecID mediumint NOT NULL AUTO\_INCREMENT comment 'The primary key that uniquely identifies the Decision/comment made',

`Date` date NOT NULL comment 'The date the decision was made on',

Comment varchar(1000) comment 'Extra information about the decision',

ApplicationID mediumint comment 'the id of the application this decision is made with regards to, if this is a decision associated with an application',

StaffID mediumint NOT NULL comment 'the staff ID of the staff member who made this decision/comment',

DecisionTypeID mediumint NOT NULL,

Reportable tinyint(1) NOT NULL comment 'a boolean that is automatically ticked if the change is deemed reportable (status changes request filled etc.)',

Sent tinyint(1) comment 'a boolean to check if the related email has been sent',

PRIMARY KEY (DecID),

INDEX (ApplicationID),

INDEX (StaffID),

INDEX (DecisionTypeID),

INDEX (Reportable),

INDEX (Sent)

)

comment='The decision/comment made for an application by a RHD staff member'

ENGINE=InnoDB;

CREATE TABLE `Research Area` (

FORCode mediumint(6) NOT NULL AUTO\_INCREMENT comment 'The Australian Field Of Research (FOR) code,

primary key,

that uniquely identifies the Research area',

Description varchar(2000) NOT NULL comment 'A small text description of the FOR i.e. 2201',

ResearchArea varchar(50) NOT NULL comment 'The FOR title; i.e. Applied Ethics',

GeneralArea varchar(50) NOT NULL comment 'The general area of the FOR',

PRIMARY KEY (FORCode),

INDEX (GeneralArea))

ENGINE=InnoDB;

CREATE TABLE `University Staff Member` (

StaffID mediumint NOT NULL AUTO\_INCREMENT comment 'The flinders uni staff ID number,

the primary key that uniquely identifies the staff member',

FName varchar(50) NOT NULL comment 'The last name of the staff member',

LName varchar(50) comment 'The first name of the staff member',

canSupervise tinyint(1) NOT NULL comment 'if the staff member is able supervise a RHD applicant',

email varchar(100) NOT NULL,

PRIMARY KEY (StaffID),

INDEX (FName)

)

comment='a university staff member who may be able to supervise a application'

ENGINE=InnoDB;

CREATE TABLE `Correspondence Method` (

CorrMethodID mediumint NOT NULL AUTO\_INCREMENT,

Method varchar(50) NOT NULL UNIQUE comment 'the method of correspondence',

PRIMARY KEY (CorrMethodID))

ENGINE=InnoDB;

CREATE TABLE `Application Status` (

ApplicationStatusID mediumint NOT NULL AUTO\_INCREMENT,

Status varchar(50) NOT NULL UNIQUE comment 'the name of the status',

Description varchar(1000) NOT NULL comment 'a full description of the status',

PRIMARY KEY (ApplicationStatusID),

UNIQUE INDEX (ApplicationStatusID)

)

comment='the possible application statuses of an application'

ENGINE=InnoDB;

CREATE TABLE `Document Type` (

DocTypeID mediumint NOT NULL AUTO\_INCREMENT,

Type varchar(50) NOT NULL UNIQUE comment 'the type of document eg. Publication,

visa etc.',

Description varchar(1000) NOT NULL comment 'a full description of the type of the document',

PRIMARY KEY (DocTypeID)

)

comment='the possible types of a document'

ENGINE=InnoDB;

CREATE TABLE `Award Type` (

AwardID mediumint NOT NULL AUTO\_INCREMENT,

Type varchar(50) NOT NULL UNIQUE comment 'the type of award sought by the applicant',

Description varchar(1000) NOT NULL comment 'a full description of the award type',

PRIMARY KEY (AwardID)

)

comment='the possible award types (degrees) sought by an application'

ENGINE=InnoDB;

CREATE TABLE Country (

CountryISOCode char(2) NOT NULL

comment 'the country corresponding ISO 3166-1 alpha-2 code',

Name varchar(50) NOT NULL UNIQUE comment 'the full name of the country',

PRIMARY KEY (CountryISOCode)

)

comment='a list of countries for reuse in nationality,

institution country,

address and visa country'

ENGINE=InnoDB;

CREATE TABLE `Supervise as` (

PrimarySupervisor tinyint(1) NOT NULL comment 'if the supervisor is a primary',

ApplicationID mediumint NOT NULL comment 'the application ID of the application the staff member will supervise',

StaffID mediumint NOT NULL comment 'the staff ID of the staff member who will supervise the applicaiton',

PRIMARY KEY (ApplicationID,

StaffID),

INDEX (ApplicationID),

INDEX (StaffID)

)

comment='the staff members who have agreed to supervise an application'

ENGINE=InnoDB;

CREATE TABLE `Application\_Research Area` (

ApplicationID mediumint NOT NULL comment 'the ID of the application',

FORCode mediumint(6) NOT NULL comment 'the FORCode research area the applicant states they want to study in',

PRIMARY KEY (ApplicationID, FORCode),

INDEX (ApplicationID),

INDEX (FORCode)

)

comment='The application and the research area they are looking to study in'

ENGINE=InnoDB;

CREATE TABLE `University Staff Member\_Research Area` (

StaffID mediumint NOT NULL comment 'the staff ID of the staff member who works in the research area',

FORCode mediumint(6) NOT NULL comment 'the field of research that the staff member works in',

PRIMARY KEY (StaffID, FORCode),

INDEX (StaffID),

INDEX (FORCode)

)

comment='the research area the staff member states they work in'

ENGINE=InnoDB;

CREATE TABLE `University Staff Member\_Research Area2` (

StaffID mediumint NOT NULL comment 'the staff ID of the staff member who oversees the research area',

FORCode mediumint(6) NOT NULL comment 'the FORCode of the research area that the staff member oversees',

PRIMARY KEY (StaffID, FORCode),

INDEX (StaffID),

INDEX (FORCode)

)

comment='the research areas that a staff member oversees (can be more than one staff member per area)'

ENGINE=InnoDB;

CREATE TABLE `University Staff Member\_Application` (

StaffID mediumint NOT NULL comment 'the staff ID of the staff member who flagged the application',

ApplicationID mediumint NOT NULL comment 'the application the staff member has flagged',

ReceiveEmailUpdates tinyint(1) NOT NULL comment 'a boolean that a user can check if they want to be alerted About an update or simply keep a reference on their application page',

PRIMARY KEY (StaffID,

ApplicationID),

INDEX (StaffID),

INDEX (ApplicationID)

)

comment='the staff members who have flagged an application to come back and check later'

ENGINE=InnoDB;

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

-- Add constraints and indexes

SELECT "Adding indexes and FK constraints" ;

ALTER TABLE Document

ADD INDEX associated (ApplicationID),

ADD CONSTRAINT associated FOREIGN KEY (ApplicationID)

REFERENCES Application (ApplicationID) ON UPDATE Cascade ON DELETE Restrict;

ALTER TABLE Document

ADD INDEX provides (ApplicantID),

ADD CONSTRAINT provides FOREIGN KEY (ApplicantID)

REFERENCES Applicant (ApplicantID) ON UPDATE Cascade ON DELETE Cascade;

ALTER TABLE Decision

ADD INDEX `specified by` (DecisionTypeID),

ADD CONSTRAINT `specified by` FOREIGN KEY (DecisionTypeID)

REFERENCES `Decision Type` (DecisionTypeID) ON UPDATE Cascade ON DELETE Restrict;

ALTER TABLE Visa

ADD INDEX `originates in` (CountryISOCode),

ADD CONSTRAINT `originates in` FOREIGN KEY (CountryISOCode)

REFERENCES Country (CountryISOCode) ON UPDATE Cascade ON DELETE Restrict;

ALTER TABLE Degree

ADD INDEX `studied in` (InstitutionCountryISOCode),

ADD CONSTRAINT `studied in` FOREIGN KEY (InstitutionCountryISOCode)

REFERENCES Country (CountryISOCode) ON UPDATE Cascade ON DELETE Restrict;

ALTER TABLE Applicant

ADD INDEX `lives in` (AddressCountryISOCode),

ADD CONSTRAINT `lives in` FOREIGN KEY (AddressCountryISOCode)

REFERENCES Country (CountryISOCode) ON UPDATE Cascade ON DELETE Restrict;

ALTER TABLE Applicant

ADD INDEX `nationality of` (NationalityCountryISOCode),

ADD CONSTRAINT `nationality of` FOREIGN KEY (NationalityCountryISOCode)

REFERENCES Country (CountryISOCode) ON UPDATE Cascade ON DELETE Restrict;

ALTER TABLE Application

ADD INDEX `will pay using` (PayMethodID),

ADD CONSTRAINT `will pay using` FOREIGN KEY (PayMethodID)

REFERENCES `Payment Method` (PayMethodID)

ON UPDATE Cascade ON DELETE Restrict;

ALTER TABLE Application

ADD INDEX submits (ApplicantID),

ADD CONSTRAINT submits FOREIGN KEY (ApplicantID)

REFERENCES Applicant (ApplicantID) ON UPDATE Cascade ON DELETE Cascade;

ALTER TABLE `Supervise as`

ADD INDEX `will supervise2` (ApplicationID),

ADD CONSTRAINT `will supervise2` FOREIGN KEY (ApplicationID)

REFERENCES Application (ApplicationID) ON UPDATE Cascade ON DELETE Cascade;

ALTER TABLE `Supervise as`

ADD INDEX `will supervise` (StaffID),

ADD CONSTRAINT `will supervise` FOREIGN KEY (StaffID)

REFERENCES `University Staff Member` (StaffID)

ON UPDATE Cascade ON DELETE Cascade;

ALTER TABLE `Application\_Research Area`

ADD INDEX `in` (ApplicationID),

ADD CONSTRAINT `in` FOREIGN KEY (ApplicationID)

REFERENCES Application (ApplicationID) ON UPDATE Cascade ON DELETE Cascade;

ALTER TABLE `Application\_Research Area`

ADD INDEX in2 (FORCode),

ADD CONSTRAINT in2 FOREIGN KEY (FORCode)

REFERENCES `Research Area` (FORCode) ON UPDATE Cascade ON DELETE Restrict;

ALTER TABLE `University Staff Member\_Research Area`

ADD INDEX `works in` (StaffID),

ADD CONSTRAINT `works in` FOREIGN KEY (StaffID)

REFERENCES `University Staff Member` (StaffID) ON UPDATE Cascade ON DELETE Cascade;

ALTER TABLE `University Staff Member\_Research Area`

ADD INDEX `works in2` (FORCode),

ADD CONSTRAINT `works in2` FOREIGN KEY (FORCode)

REFERENCES `Research Area` (FORCode) ON UPDATE Cascade ON DELETE Restrict;

ALTER TABLE `University Staff Member\_Research Area2`

ADD INDEX Oversees (StaffID),

ADD CONSTRAINT Oversees FOREIGN KEY (StaffID)

REFERENCES `University Staff Member` (StaffID) ON UPDATE Cascade ON DELETE Restrict;

ALTER TABLE `University Staff Member\_Research Area2`

ADD INDEX Oversees2 (FORCode),

ADD CONSTRAINT Oversees2 FOREIGN KEY (FORCode)

REFERENCES `Research Area` (FORCode) ON UPDATE Cascade ON DELETE Cascade;

ALTER TABLE Decision

ADD INDEX Decides (ApplicationID),

ADD CONSTRAINT Decides FOREIGN KEY (ApplicationID)

REFERENCES Application (ApplicationID) ON UPDATE Cascade ON DELETE Cascade;

ALTER TABLE Decision

ADD INDEX Decides2 (StaffID),

ADD CONSTRAINT Decides2 FOREIGN KEY (StaffID)

REFERENCES `University Staff Member` (StaffID)

ON UPDATE Cascade ON DELETE Restrict;

ALTER TABLE Visa

ADD INDEX `categorised by` (VisaStatusID),

ADD CONSTRAINT `categorised by` FOREIGN KEY (VisaStatusID)

REFERENCES `Visa Status` (VisaStatusID) ON UPDATE Cascade ON DELETE Restrict;

ALTER TABLE Application

ADD INDEX `defined by` (applicationStatusID),

ADD CONSTRAINT `defined by` FOREIGN KEY (applicationStatusID)

REFERENCES `Application Status` (ApplicationStatusID)

ON UPDATE Cascade ON DELETE Restrict;

ALTER TABLE Document

ADD INDEX `has a` (DocStatusID),

ADD CONSTRAINT `has a` FOREIGN KEY (DocStatusID)

REFERENCES `Document Status` (DocStatusID)

ON UPDATE Cascade ON DELETE Restrict;

ALTER TABLE `University Staff Member\_Application`

ADD INDEX flags (StaffID),

ADD CONSTRAINT flags FOREIGN KEY (StaffID)

REFERENCES `University Staff Member` (StaffID)

ON UPDATE Cascade ON DELETE Cascade;

ALTER TABLE `University Staff Member\_Application`

ADD INDEX flags2 (ApplicationID),

ADD CONSTRAINT flags2 FOREIGN KEY (ApplicationID)

REFERENCES Application (ApplicationID) ON UPDATE Cascade ON DELETE Cascade;

ALTER TABLE Referee

ADD INDEX `supported by` (ApplicationID),

ADD CONSTRAINT `supported by` FOREIGN KEY (ApplicationID)

REFERENCES Application (ApplicationID) ON UPDATE Cascade ON DELETE Cascade;

ALTER TABLE Correspondence

ADD INDEX `corresponds with` (ApplicationID),

ADD CONSTRAINT `corresponds with` FOREIGN KEY (ApplicationID)

REFERENCES Application (ApplicationID) ON UPDATE Cascade ON DELETE Cascade;

ALTER TABLE Correspondence

ADD INDEX `corresponds with2` (StaffID),

ADD CONSTRAINT `corresponds with2` FOREIGN KEY (StaffID)

REFERENCES `University Staff Member` (StaffID)

ON UPDATE Cascade ON DELETE Restrict;

ALTER TABLE Document

ADD INDEX `of` (DocTypeID),

ADD CONSTRAINT `of` FOREIGN KEY (DocTypeID)

REFERENCES `Document Type` (DocTypeID) ON UPDATE Cascade ON DELETE Restrict;

ALTER TABLE Application

ADD INDEX Manages (ManagedByStaffID),

ADD CONSTRAINT Manages FOREIGN KEY (ManagedByStaffID)

REFERENCES `University Staff Member` (StaffID)

ON UPDATE Cascade ON DELETE Set null;

ALTER TABLE Degree

ADD INDEX holds (ApplicantID),

ADD CONSTRAINT holds FOREIGN KEY (ApplicantID)

REFERENCES Applicant (ApplicantID) ON UPDATE Cascade ON DELETE Cascade;

ALTER TABLE Publication

ADD INDEX authored (ApplicantID),

ADD CONSTRAINT authored FOREIGN KEY (ApplicantID)

REFERENCES Applicant (ApplicantID) ON UPDATE Cascade ON DELETE Cascade;

ALTER TABLE Visa

ADD INDEX `may have` (ApplicantID),

ADD CONSTRAINT `may have` FOREIGN KEY (ApplicantID)

REFERENCES Applicant (ApplicantID) ON UPDATE Cascade ON DELETE Cascade;

ALTER TABLE Application

ADD INDEX seeks (awardID),

ADD CONSTRAINT seeks FOREIGN KEY (awardID)

REFERENCES `Award Type` (AwardID) ON UPDATE Cascade ON DELETE Restrict;

ALTER TABLE Application

ADD INDEX `last to update` (LastModifiedByStaffID),

ADD CONSTRAINT `last to update` FOREIGN KEY (LastModifiedByStaffID)

REFERENCES `University Staff Member` (StaffID)

ON UPDATE Cascade ON DELETE Restrict;

ALTER TABLE Applicant

ADD INDEX `last to modify` (LastModifiedByStaffID),

ADD CONSTRAINT `last to modify` FOREIGN KEY (LastModifiedByStaffID)

REFERENCES `University Staff Member` (StaffID)

ON UPDATE Cascade ON DELETE Restrict;

ALTER TABLE Correspondence

ADD INDEX `using` (CorrMethodID),

ADD CONSTRAINT `using` FOREIGN KEY (CorrMethodID)

REFERENCES `Correspondence Method` (CorrMethodID)

ON UPDATE Cascade ON DELETE Restrict;

-- Load up other parts

source create\_views.sql

source security.sql

source create\_triggers.sql

## Create views

### Utilities

SELECT 'Creating utility views';

-- A utility view with lookup values cached out

DROP VIEW IF EXISTS Application\_Expanded ;

CREATE VIEW Application\_Expanded AS

SELECT Appt.FName, Appt.LName, Appt.Email, App.\*, AppStat.Status

FROM Application App

INNER JOIN Applicant Appt

ON Appt.ApplicantID = App.ApplicantID

INNER JOIN `Application Status` AppStat

ON AppStat.ApplicationStatusID = App.applicationStatusID;

DROP VIEW IF EXISTS Application\_Ongoing\_Expanded ;

CREATE VIEW Application\_Ongoing\_Expanded AS

SELECT \*

FROM Application\_Expanded App

WHERE App.Status LIKE 'ongoing%';

-- Friendly view of application status

DROP VIEW IF EXISTS `Application\_Quickview` ;

CREATE VIEW `Application\_Quickview` AS

SELECT

ApplicationID,

FName,

LName,

Email,

Status,

AddressConfirmed,

DegreeConfirmed,

VisaStatusConfirmed,

ProposalConfirmed,

HasResearchAreas,

HasPrimarySuper,

PayMethConfirmed,

EngProfConfirmed,

RefereesConfirmed

FROM Application\_Expanded

GROUP BY ApplicationID

ORDER BY LName;

-- Create views to separate primary supervisors from associate

DROP VIEW IF EXISTS `Supervise as primary` ;

CREATE VIEW `Supervise as primary` AS

SELECT \*

FROM `Supervise as` AS Super

WHERE Super.PrimarySupervisor = 1;

DROP VIEW IF EXISTS `Supervise as associate` ;

CREATE VIEW `Supervise as associate` AS

SELECT \*

FROM `Supervise as` AS Super

WHERE Super.PrimarySupervisor = 0;

DELIMITER $$

-- A function to simulate user switching, as we don’t have facility to do this

-- for real on the university system

-- +--------+--------+----------+--------------+-------------------------------+

-- |StaffID | FName | LName | canSupervise | email |

-- +--------+--------+----------+--------------+-------------------------------+

-- | 1000 | Denise | de Vries | 1 | denise.deries@flinders.edu.au |

-- | 1001 | Paul | Calder | 1 | paul.calder@flinders.edu.au |

-- | 1002 | John | Roddick | 1 | john.roddick@flinders.edu.au |

-- | 1003 | Jennie | Brand | 0 | jennie.brand@flinders.edu.au |

-- +--------+--------+----------+--------------+-------------------------------+

DROP FUNCTION IF EXISTS CURRENT\_RHD\_USER $$

CREATE FUNCTION CURRENT\_RHD\_USER()

RETURNS mediumint

DETERMINISTIC

BEGIN

RETURN 1000;

-- RETURN 1001;

END $$

DROP FUNCTION IF EXISTS CURRENT\_RHD\_TIMESTAMP $$

CREATE FUNCTION CURRENT\_RHD\_TIMESTAMP()

RETURNS char(19)

DETERMINISTIC

BEGIN

RETURN '2014-05-17 12:00:00';

-- RETURN current\_timestamp();

END $$

DROP FUNCTION IF EXISTS CURRENT\_RHD\_DATE $$

CREATE FUNCTION CURRENT\_RHD\_DATE()

RETURNS char(10)

DETERMINISTIC

BEGIN

RETURN DATE(CURRENT\_RHD\_TIMESTAMP());

END $$

DELIMITER ;

### Views for all staff, including professional staff

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

-- For a member of staff, create a view that's shows all the ongoing

-- applications that they're in some way working on.

--

-- In the following, we have hard-coded a user/staff ID of 1000, as we don't

-- have permission in the CSEM MySQL server to create database users. In a real

-- system, we would replace the hard-coded values with the function USER() to

-- get the current database user ID.

SELECT 'Creating views for professional/all staff';

-- All the Applications that the current user is marked as primary supervisor

DROP VIEW IF EXISTS `MyRHDApps\_SupervisedPrimaryByMe\_Expanded` ;

CREATE VIEW `MyRHDApps\_SupervisedPrimaryByMe\_Expanded` AS

SELECT 'a) primary supervisor' AS `My role`, App.\*

FROM Application\_Ongoing\_Expanded App

INNER JOIN `Supervise as primary` Super

ON Super.ApplicationID = App.ApplicationID

WHERE Super.StaffID = CURRENT\_RHD\_USER() ;

-- ... as associate supervisor

DROP VIEW IF EXISTS `MyRHDApps\_SupervisedAssociateByMe\_Expanded` ;

CREATE VIEW `MyRHDApps\_SupervisedAssociateByMe\_Expanded` AS

SELECT 'b) associate supervisor' AS `My role`, App.\*

FROM Application\_Ongoing\_Expanded App

INNER JOIN `Supervise as associate` Super

ON Super.ApplicationID = App.ApplicationID

WHERE Super.StaffID = CURRENT\_RHD\_USER() ;

-- All the Applications that the current user has flagged

DROP VIEW IF EXISTS `MyRHDApps\_FlaggedByMe\_Expanded` ;

CREATE VIEW `MyRHDApps\_FlaggedByMe\_Expanded` AS

SELECT 'c) flagged by me' AS `My role`, App.\*

FROM Application\_Ongoing\_Expanded App

INNER JOIN `University Staff Member\_Application` Flag

ON Flag.ApplicationID = App.ApplicationID

WHERE Flag.StaffID = CURRENT\_RHD\_USER()

ORDER BY App.ApplicationID DESC;

-- All the Applications that the current user has most recently modified

DROP VIEW IF EXISTS MyRHDApps\_LastModifiedByMe\_Expanded ;

CREATE VIEW MyRHDApps\_LastModifiedByMe\_Expanded AS

SELECT 'd) modified by me' AS `My role`, App.\*

FROM Application\_Ongoing\_Expanded App

WHERE App.LastModifiedByStaffID = CURRENT\_RHD\_USER()

ORDER BY App.DateLastModified;

-- Composite view of the above

DROP VIEW IF EXISTS `MyRHDApps\_Expanded` ;

CREATE VIEW `MyRHDApps\_Expanded` AS

SELECT \* FROM MyRHDApps\_SupervisedPrimaryByMe\_Expanded

UNION

SELECT \* FROM MyRHDApps\_SupervisedAssociateByMe\_Expanded

UNION

SELECT \* FROM MyRHDApps\_FlaggedByMe\_Expanded

UNION

SELECT \* FROM MyRHDApps\_LastModifiedByMe\_Expanded;

-- As above, but with irrelevent columns suppressed

DROP VIEW IF EXISTS `MyRHDApps` ;

CREATE VIEW `MyRHDApps` AS

SELECT

`My role`,

FName,

LName,

Email,

Status,

AddressConfirmed,

DegreeConfirmed,

VisaStatusConfirmed,

ProposalConfirmed,

HasResearchAreas,

HasPrimarySuper,

PayMethConfirmed,

EngProfConfirmed,

RefereesConfirmed

FROM MyRHDApps\_Expanded

GROUP BY ApplicationID

ORDER BY `My role`, LName;

### Views for academic staff

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

-- Create views specific to academic staff members

SELECT 'Creating views for academic staff';

-- List all the recent applications in the same research area as the current

-- user has nominated an interest in

DROP VIEW IF EXISTS Recent\_Applications\_In\_My\_Research\_Area;

CREATE VIEW Recent\_Applications\_In\_My\_Research\_Area AS

SELECT

App.DateAdded,

App.ApplicationID,

Appt.FName,

Appt.LName,

Appt.Email,

RA.Description AS `Research Area Description`,

USM.FName AS StaffInResearchAreaFName,

USM.LName AS StaffInResearchAreaLName

FROM Application App

INNER JOIN Applicant Appt

ON App.ApplicantID = Appt.ApplicantID

INNER JOIN `Application\_Research Area` ARA

ON App.ApplicationID = ARA.ApplicationID

INNER JOIN `Research Area` AS RA

ON ARA.FORCode = RA.FORCode

INNER JOIN `University Staff Member\_Research Area` USM\_RA

ON ARA.FORCode = USM\_RA.FORCode

INNER JOIN `University Staff Member` USM

ON USM\_RA.StaffID = USM.StaffID

WHERE USM\_RA.StaffID = CURRENT\_RHD\_USER()

AND DATEDIFF(CURRENT\_RHD\_DATE(), App.DateAdded) >= 7

ORDER BY App.DateAdded;

### Views for RHD administration staff

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

-- Views for RHD staff.

SELECT 'Creating views for RHD staff';

-- All ongoing applications and the contact staff

DROP VIEW IF EXISTS Application\_Staff\_Overview ;

CREATE VIEW Application\_Staff\_Overview AS

SELECT

AppExpand.ApplicationID,

AppExpand.FName,

AppExpand.LName,

AppExpand.Email,

PrimaryUSM.FName AS PrimarySupervisorFName,

PrimaryUSM.LName AS PrimarySupervisorLName,

COUNT(AssociateSuper.StaffID) AS `Associate supervisor count`,

LastModifiedByUSM.FName AS LastModifiedByFName,

LastModifiedByUSM.LName AS LastModifiedByLName

FROM Application\_Ongoing\_Expanded AS AppExpand

LEFT JOIN (`Supervise as primary` AS PrimarySuper,

`University Staff Member` AS PrimaryUSM)

ON (AppExpand.ApplicationID = PrimarySuper.ApplicationID

AND PrimarySuper.StaffID = PrimaryUSM.StaffID)

LEFT JOIN (`Supervise as associate` AS AssociateSuper)

ON (AppExpand.ApplicationID = AssociateSuper.ApplicationID)

LEFT JOIN (`University Staff Member` AS LastModifiedByUSM)

ON (AppExpand.LastModifiedByStaffID = LastModifiedByUSM.StaffID)

GROUP BY AppExpand.ApplicationID

ORDER BY LName;

-- a utility view for the view below

DROP VIEW IF EXISTS Application\_Primary\_Supervisor ;

CREATE VIEW Application\_Primary\_Supervisor AS

SELECT

App.\*,

Super.StaffID,

Super.PrimarySupervisor,

SUM(Super.PrimarySupervisor) AS PriSuperSum

FROM Application\_Ongoing\_Expanded App

LEFT OUTER JOIN `Supervise as` Super

ON Super.ApplicationID = App.ApplicationID

GROUP BY App.ApplicationID;

-- List only those ongoing applications that don't yet have a primary supervisor

DROP VIEW IF EXISTS Application\_Without\_Supervisor\_Inner ;

CREATE VIEW Application\_Without\_Supervisor\_Inner AS

SELECT

AppPri.ApplicationID,

AppPri.FName,

AppPri.LName,

AppPri.Email,

AppPri.DateAdded,

COUNT(AssociateSuper.StaffID) AS `Associate supervisor count`

FROM Application\_Primary\_Supervisor AS AppPri

LEFT JOIN (`Supervise as associate` AS AssociateSuper)

ON (AppPri.ApplicationID = AssociateSuper.ApplicationID)

WHERE AppPri.PriSuperSum <> 1

OR AppPri.PriSuperSum IS NULL

GROUP BY AppPri.ApplicationID

ORDER BY DateAdded ;

-- Show those applications that don't yet have a primary supervisor.

-- Where the application has nominated a research area, report which staff

-- member oversees that research area.

DROP VIEW IF EXISTS Application\_Without\_Supervisor ;

CREATE VIEW Application\_Without\_Supervisor AS

SELECT

AppWoSuper.\*,

AppRA.FORCode,

USM.FName as StaffOverseeingAreaFName,

USM.LName as StaffOverseeingAreaLName

FROM Application\_Without\_Supervisor\_Inner AppWoSuper

LEFT JOIN (`Application\_Research Area` AS AppRA)

ON (AppWoSuper.ApplicationID = AppRA.ApplicationID)

LEFT JOIN (`Research Area` AS RA)

ON (AppRA.FORCode = RA.FORCode)

LEFT JOIN (`University Staff Member\_Research Area2` AS USM\_RA\_Oversees)

ON (RA.FORCode = USM\_RA\_Oversees.FORCode)

LEFT JOIN (`University Staff Member` AS USM)

ON (USM\_RA\_Oversees.StaffID = USM.StaffID)

ORDER BY DateAdded;

## Create procedure for setting user privileges

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

-- security.sql

--

-- Script to assist DB admins to enforce security policies. This needs to be

-- read into the DB with admin privileges.

--

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

DROP PROCEDURE IF EXISTS insert\_new\_staff ;

DELIMITER $$

CREATE PROCEDURE insert\_new\_staff(

IN p\_staffID int(10),

IN p\_password varchar(255),

IN p\_fName varchar(50),

IN p\_lName varchar(50),

IN p\_canSupervise int(1),

IN p\_email varchar(100) )

BEGIN

-- Create a database user.

-- Have to use dynamic SQL here because of the password.

SET @create\_user\_query = CONCAT(

'CREATE USER "', p\_staffID, '" IDENTIFIED BY "', p\_password, '" ');

PREPARE create\_user\_stmt FROM @create\_user\_query ;

EXECUTE create\_user\_stmt ;

DEALLOCATE PREPARE create\_user\_stmt;

-- add the staff member

INSERT INTO `University Staff Member` (StaffID, FName, LName, canSupervise,

Email)

VALUES (p\_staffID, p\_fName, p\_lName, p\_canSupervise, p\_email) ;

-- Set the appropriate permissions:

-- - no delete permissions

-- - only select on 'University Staff Member'

-- - select, update and insert on all other tables

GRANT SELECT ON TABLE `University Staff Member` TO p\_staffID;

GRANT SELECT,UPDATE,INSERT ON TABLE `Applicant` TO p\_staffID;

GRANT SELECT,UPDATE,INSERT ON TABLE `Application` TO p\_staffID;

GRANT SELECT,UPDATE,INSERT ON TABLE `Application Status` TO p\_staffID;

GRANT SELECT,UPDATE,INSERT ON TABLE `Application\_Research Area` TO p\_staffID;

GRANT SELECT,UPDATE,INSERT ON TABLE `Award Type` TO p\_staffID;

GRANT SELECT,UPDATE,INSERT ON TABLE `Correspondence` TO p\_staffID;

GRANT SELECT,UPDATE,INSERT ON TABLE `Correspondence Method` TO p\_staffID;

GRANT SELECT,UPDATE,INSERT ON TABLE `Country` TO p\_staffID;

GRANT SELECT,UPDATE,INSERT ON TABLE `Decision` TO p\_staffID;

GRANT SELECT,UPDATE,INSERT ON TABLE `Decision Type` TO p\_staffID;

GRANT SELECT,UPDATE,INSERT ON TABLE `Degree` TO p\_staffID;

GRANT SELECT,UPDATE,INSERT ON TABLE `Document` TO p\_staffID;

GRANT SELECT,UPDATE,INSERT ON TABLE `Document Status` TO p\_staffID;

GRANT SELECT,UPDATE,INSERT ON TABLE `Document Type` TO p\_staffID;

GRANT SELECT,UPDATE,INSERT ON TABLE `Payment Method` TO p\_staffID;

GRANT SELECT,UPDATE,INSERT ON TABLE `Publication` TO p\_staffID;

GRANT SELECT,UPDATE,INSERT ON TABLE `Referee` TO p\_staffID;

GRANT SELECT,UPDATE,INSERT ON TABLE `Research Area` TO p\_staffID;

GRANT SELECT,UPDATE,INSERT ON TABLE `Visa` TO p\_staffID;

GRANT SELECT,UPDATE,INSERT ON TABLE `Visa Status` TO p\_staffID;

-- These tables consist (almost) entirely of FKs, so update isn't really

-- needed, and this makes monitoring changes to these with triggers more

-- straightforward.

GRANT SELECT,INSERT,DELETE ON TABLE `Supervise as` TO p\_staffID;

GRANT SELECT,INSERT,DELETE ON TABLE `University Staff Member\_Application`

TO p\_staffID;

GRANT SELECT,INSERT,DELETE ON TABLE `University Staff Member\_Research Area`

TO p\_staffID;

GRANT SELECT,INSERT,DELETE ON TABLE `University Staff Member\_Research Area2`

TO p\_staffID;

END $$

DELIMITER ;

## Create triggers for tracking changes and enforcing constraints

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

-- Create triggers for logging and constraint checking

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

-- Make sure there's only ever one primary supervisor for each application

DELIMITER $$

DROP TRIGGER IF EXISTS checkPrimSuper $$

CREATE TRIGGER checkPrimSuper BEFORE INSERT ON `Supervise as`

FOR EACH ROW

BEGIN

DECLARE existingCount int default 0;

SELECT count(\*)

FROM `Supervise as`

WHERE PrimarySupervisor>0

AND ApplicationID=NEW.ApplicationID

INTO existingCount;

IF existingCount IS NOT NULL THEN

IF existingCount > 0 THEN

SET NEW.PrimarySupervisor = 0;

END IF;

END IF;

END $$

DELIMITER ;

DELIMITER $$

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

-- Record changes to Research Areas to email these to the change subject later

DROP TRIGGER IF EXISTS USM\_RESEARCHAREA\_AI $$

CREATE TRIGGER USM\_RESEARCHAREA\_AI AFTER INSERT ON

`University Staff Member\_Research Area`

FOR EACH ROW

BEGIN

DECLARE v\_forCodeDescription varchar(2000) ;

DECLARE v\_comment varchar(1000) ;

DECLARE v\_agentFName varchar(50) ;

DECLARE v\_agentLName varchar(50) ;

DECLARE v\_decisionTypeID mediumint(9) ;

SELECT Description

FROM `Research Area` RA

WHERE RA.FORCode = NEW.FORCode

INTO v\_forCodeDescription ;

SELECT FName, LName

FROM `University Staff Member` AS USM

WHERE USM.StaffID = CURRENT\_RHD\_USER()

INTO v\_agentFName, v\_agentLName;

SELECT DecisionTypeID

FROM `Decision Type` AS DT

WHERE DT.type = 'change.research\_area.addition'

INTO v\_decisionTypeID;

SET v\_comment = CONCAT('A new Field Of Research Code ', NEW.FORCode,

' - ''', SUBSTRING(v\_forCodeDescription, 1, 800),

''' has been associated to your RHD account by ', v\_agentFName, ' ',

v\_agentLName);

INSERT INTO Decision(Date, Comment, StaffID, DecisionTypeID, Reportable, Sent)

VALUES (CURRENT\_DATE(), v\_comment, NEW.StaffID, v\_decisionTypeID, 1, 0);

END $$

DELIMITER ;

DELIMITER $$

DROP TRIGGER IF EXISTS USM\_RESEARCHAREA\_AD $$

CREATE TRIGGER USM\_RESEARCHAREA\_AD AFTER DELETE ON

`University Staff Member\_Research Area`

FOR EACH ROW

BEGIN

DECLARE v\_forCodeDescription varchar(2000) ;

DECLARE v\_comment varchar(1000) ;

DECLARE v\_agentFName varchar(50) ;

DECLARE v\_agentLName varchar(50) ;

DECLARE v\_decisionTypeID mediumint(9) ;

SELECT Description

FROM `Research Area` RA

WHERE RA.FORCode = OLD.FORCode

INTO v\_forCodeDescription ;

SELECT FName, LName

FROM `University Staff Member` AS USM

WHERE USM.StaffID = CURRENT\_RHD\_USER()

INTO v\_agentFName, v\_agentLName;

SELECT DecisionTypeID

FROM `Decision Type` AS DT

WHERE DT.type = 'change.research\_area.deletion'

INTO v\_decisionTypeID;

SET v\_comment = CONCAT('An existing Field Of Research Code ', OLD.FORCode,

' - ''', SUBSTRING(v\_forCodeDescription, 1, 800),

''' has been removed from association with your RHD account by ',

v\_agentFName, ' ', v\_agentLName);

INSERT INTO Decision(Date, Comment, StaffID, DecisionTypeID, Reportable, Sent)

VALUES (CURRENT\_DATE(), v\_comment, OLD.StaffID, v\_decisionTypeID, 1, 0);

END $$

DELIMITER ;

DELIMITER $$

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

-- Record changes to Research Areas to email these to the change subject later

DROP TRIGGER IF EXISTS USM\_RESEARCHAREA2\_AI $$

CREATE TRIGGER USM\_RESEARCHAREA2\_AI AFTER INSERT ON

`University Staff Member\_Research Area2`

FOR EACH ROW

BEGIN

DECLARE v\_forCodeDescription varchar(2000) ;

DECLARE v\_comment varchar(1000) ;

DECLARE v\_agentFName varchar(50) ;

DECLARE v\_agentLName varchar(50) ;

DECLARE v\_decisionTypeID mediumint(9) ;

SELECT Description

FROM `Research Area` RA

WHERE RA.FORCode = NEW.FORCode

INTO v\_forCodeDescription ;

SELECT FName, LName

FROM `University Staff Member` AS USM

WHERE USM.StaffID = CURRENT\_RHD\_USER()

INTO v\_agentFName, v\_agentLName;

SELECT DecisionTypeID

FROM `Decision Type` AS DT

WHERE DT.type LIKE 'change.research\_area\_oversees.addition'

INTO v\_decisionTypeID;

SET v\_comment = CONCAT(

'You have been registered as overseeing Field Of Research area ', NEW.FORCode,

' - ''', SUBSTRING(v\_forCodeDescription, 1, 800),

''', this change made by ', v\_agentFName, ' ',

v\_agentLName);

INSERT INTO Decision(Date, Comment, StaffID, DecisionTypeID, Reportable, Sent)

VALUES (CURRENT\_DATE(), v\_comment, NEW.StaffID, v\_decisionTypeID, 1, 0);

END $$

DELIMITER ;

DELIMITER $$

DROP TRIGGER IF EXISTS USM\_RESEARCHAREA2\_AD $$

CREATE TRIGGER USM\_RESEARCHAREA2\_AD AFTER DELETE ON

`University Staff Member\_Research Area2`

FOR EACH ROW

BEGIN

DECLARE v\_forCodeDescription varchar(2000) ;

DECLARE v\_comment varchar(1000) ;

DECLARE v\_agentFName varchar(50) ;

DECLARE v\_agentLName varchar(50) ;

DECLARE v\_decisionTypeID mediumint(9) ;

SELECT Description

FROM `Research Area` RA

WHERE RA.FORCode = OLD.FORCode

INTO v\_forCodeDescription ;

SELECT FName, LName

FROM `University Staff Member` AS USM

WHERE USM.StaffID = CURRENT\_RHD\_USER()

INTO v\_agentFName, v\_agentLName;

SELECT DecisionTypeID

FROM `Decision Type` AS DT

WHERE DT.type = 'change.research\_area\_oversees.deletion'

INTO v\_decisionTypeID;

SET v\_comment = CONCAT(

'You have been deregistered as overseeing Field Of Research area ',

OLD.FORCode, ' - ''', SUBSTRING(v\_forCodeDescription, 1, 800),

''', this change made by ', v\_agentFName, ' ', v\_agentLName);

INSERT INTO Decision(Date, Comment, StaffID, DecisionTypeID, Reportable, Sent)

VALUES (CURRENT\_DATE(), v\_comment, OLD.StaffID, v\_decisionTypeID, 1, 0);

END $$

DELIMITER ;

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

-- Add a triggers to record changes to supervisors

DELIMITER $$

DROP TRIGGER IF EXISTS SUPERVISE\_AS\_AI $$

CREATE TRIGGER SUPERVISE\_AS\_AI AFTER INSERT ON

`Supervise as`

FOR EACH ROW

BEGIN

DECLARE v\_apptFName varchar(50) ;

DECLARE v\_apptLName varchar(50) ;

DECLARE v\_apptEmail varchar(100) ;

DECLARE v\_agentFName varchar(50) ;

DECLARE v\_agentLName varchar(50) ;

DECLARE v\_decisionTypeID mediumint(9) ;

DECLARE v\_supervisionRole varchar(20) DEFAULT 'an associate';

DECLARE v\_comment varchar(1000) ;

SELECT Appt.FName, Appt.LName, Appt.Email

FROM Application App

INNER JOIN Applicant Appt ON (App.ApplicantID = Appt.ApplicantID)

WHERE App.ApplicationID = NEW.ApplicationID

INTO v\_apptFName, v\_apptLName, v\_apptEmail ;

IF v\_apptLName IS NULL THEN

SET v\_apptLName = ' (no lastname registered)';

END IF;

IF v\_apptEmail IS NULL THEN

SET v\_apptEmail = ' (no email registered)';

END IF;

SELECT FName, LName

FROM `University Staff Member` AS USM

WHERE USM.StaffID = CURRENT\_RHD\_USER()

INTO v\_agentFName, v\_agentLName;

SELECT DecisionTypeID

FROM `Decision Type` AS DT

WHERE DT.type LIKE 'change.supervisor.addition'

INTO v\_decisionTypeID;

IF NEW.PrimarySupervisor = 1 THEN

SET v\_supervisionRole = 'the primary';

END IF;

SET v\_comment = CONCAT(

'You have been registered as ', v\_supervisionRole,

' supervisor for an RHD application from ', v\_apptFName, ' ', v\_apptLName,

' (application ID ', NEW.ApplicationID, '). This change made by ',

v\_agentFName, ' ', v\_agentLName, '.');

INSERT INTO Decision(Date, Comment, StaffID, DecisionTypeID, Reportable, Sent)

VALUES (CURRENT\_DATE(), v\_comment, NEW.StaffID, v\_decisionTypeID, 1, 0);

END $$

DELIMITER ;

DELIMITER $$

DROP TRIGGER IF EXISTS SUPERVISE\_AS\_AD $$

CREATE TRIGGER SUPERVISE\_AS\_AD AFTER DELETE ON

`Supervise as`

FOR EACH ROW

BEGIN

DECLARE v\_apptFName varchar(50) ;

DECLARE v\_apptLName varchar(50) ;

DECLARE v\_apptEmail varchar(100) ;

DECLARE v\_agentFName varchar(50) ;

DECLARE v\_agentLName varchar(50) ;

DECLARE v\_decisionTypeID mediumint(9) ;

DECLARE v\_supervisionRole varchar(20) DEFAULT 'an associate';

DECLARE v\_comment varchar(1000) ;

SELECT Appt.FName, Appt.LName, Appt.Email

FROM Application App

INNER JOIN Applicant Appt ON (App.ApplicantID = Appt.ApplicantID)

WHERE App.ApplicationID = OLD.ApplicationID

INTO v\_apptFName, v\_apptLName, v\_apptEmail ;

IF v\_apptLName IS NULL THEN

SET v\_apptLName = ' (no lastname registered)';

END IF;

IF v\_apptEmail IS NULL THEN

SET v\_apptEmail = ' (no email registered)';

END IF;

SELECT FName, LName

FROM `University Staff Member` AS USM

WHERE USM.StaffID = CURRENT\_RHD\_USER()

INTO v\_agentFName, v\_agentLName;

SELECT DecisionTypeID

FROM `Decision Type` AS DT

WHERE DT.type LIKE 'change.supervisor.addition'

INTO v\_decisionTypeID;

IF OLD.PrimarySupervisor = 1 THEN

SET v\_supervisionRole = 'the primary';

END IF;

SET v\_comment = CONCAT(

'You have been deregistered as ', v\_supervisionRole,

' supervisor for an RHD application from ', v\_apptFName, ' ', v\_apptLName,

' (application ID ', OLD.ApplicationID, '). This change made by ',

v\_agentFName, ' ', v\_agentLName, '.');

INSERT INTO Decision(Date, Comment, StaffID, DecisionTypeID, Reportable, Sent)

VALUES (CURRENT\_DATE(), v\_comment, OLD.StaffID, v\_decisionTypeID, 1, 0);

END $$

DELIMITER ;

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

-- Record changes to 'flagging' an application.

-- Adding a flag

DELIMITER $$

DROP TRIGGER IF EXISTS USM\_APPLICATION\_AI $$

CREATE TRIGGER USM\_APPLICATION\_AI AFTER INSERT ON

`University Staff Member\_Application`

FOR EACH ROW

BEGIN

DECLARE v\_apptFName varchar(50) ;

DECLARE v\_apptLName varchar(50) ;

DECLARE v\_apptEmail varchar(100) ;

DECLARE v\_agentFName varchar(50) ;

DECLARE v\_agentLName varchar(50) ;

DECLARE v\_decisionTypeID mediumint(9) ;

DECLARE v\_emailUpdates varchar(100) DEFAULT ' not';

DECLARE v\_comment varchar(1000) ;

SELECT Appt.FName, Appt.LName, Appt.Email

FROM Application App

INNER JOIN Applicant Appt ON (App.ApplicantID = Appt.ApplicantID)

WHERE App.ApplicationID = NEW.ApplicationID

INTO v\_apptFName, v\_apptLName, v\_apptEmail ;

IF v\_apptLName IS NULL THEN

SET v\_apptLName = ' (no lastname registered)';

END IF;

IF v\_apptEmail IS NULL THEN

SET v\_apptEmail = ' (no email registered)';

END IF;

SELECT FName, LName

FROM `University Staff Member` AS USM

WHERE USM.StaffID = CURRENT\_RHD\_USER()

INTO v\_agentFName, v\_agentLName;

SELECT DecisionTypeID

FROM `Decision Type` AS DT

WHERE DT.type LIKE 'change.flag.addition'

INTO v\_decisionTypeID;

IF NEW.ReceiveEmailUpdates = 1 THEN

SET v\_emailUpdates = '';

END IF;

SET v\_comment = CONCAT(

'An flag for you on an RHD application from ',

v\_apptFName, ' ', v\_apptLName,

' (application ID ', NEW.ApplicationID, ') has been added. ',

'You are', v\_emailUpdates,

' registered for email updates to this application. ',

'This change made by ', v\_agentFName, ' ', v\_agentLName, '.');

INSERT INTO Decision(Date, Comment, StaffID, DecisionTypeID, Reportable, Sent)

VALUES (CURRENT\_DATE(), v\_comment, NEW.StaffID, v\_decisionTypeID, 1, 0);

END $$

DELIMITER ;

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

-- Record changes to 'flagging' an application.

-- Removing a flag.

DELIMITER $$

DROP TRIGGER IF EXISTS USM\_APPLICATION\_AD $$

CREATE TRIGGER USM\_APPLICATION\_AD AFTER DELETE ON

`University Staff Member\_Application`

FOR EACH ROW

BEGIN

DECLARE v\_apptFName varchar(50) ;

DECLARE v\_apptLName varchar(50) DEFAULT ' (no lastname registered)';

DECLARE v\_apptEmail varchar(100) DEFAULT ' (no email registered)';

DECLARE v\_agentFName varchar(50) ;

DECLARE v\_agentLName varchar(50) DEFAULT ' (no lastname registered)';

DECLARE v\_decisionTypeID mediumint(9) ;

DECLARE v\_comment varchar(1000) ;

SELECT Appt.FName, Appt.LName, Appt.Email

FROM Application App

INNER JOIN Applicant Appt ON (App.ApplicantID = Appt.ApplicantID)

WHERE App.ApplicationID = OLD.ApplicationID

INTO v\_apptFName, v\_apptLName, v\_apptEmail ;

IF v\_apptLName IS NULL THEN

SET v\_apptLName = ' (no lastname registered)';

END IF;

IF v\_apptEmail IS NULL THEN

SET v\_apptEmail = ' (no email registered)';

END IF;

SELECT FName, LName

FROM `University Staff Member` AS USM

WHERE USM.StaffID = CURRENT\_RHD\_USER()

INTO v\_agentFName, v\_agentLName;

SELECT DecisionTypeID

FROM `Decision Type` AS DT

WHERE DT.type LIKE 'change.flag.addition'

INTO v\_decisionTypeID;

SET v\_comment = CONCAT(

'Your flagged interest in an RHD application from ',

v\_apptFName, ' ', v\_apptLName,

' (application ID ', OLD.ApplicationID, ') has been removed. ',

'This change made by ', v\_agentFName, ' ', v\_agentLName, '.');

INSERT INTO Decision(Date, Comment, StaffID, DecisionTypeID, Reportable, Sent)

VALUES (CURRENT\_DATE(), v\_comment, OLD.StaffID, v\_decisionTypeID, 1, 0);

END $$

DELIMITER ;

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

-- Applicant emails are automatically inserted into new Applications

DELIMITER $$

DROP TRIGGER IF EXISTS APPLICATION\_EMAIL\_INSERT $$

CREATE TRIGGER APPLICATION\_EMAIL\_INSERT BEFORE INSERT ON `Application`

FOR EACH ROW

BEGIN

SET NEW.ApplicantEmail=(SELECT Email FROM Applicant WHERE ApplicantID=New.ApplicantID);

END $$

DELIMITER ;

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

-- UPDATE of Applicant emails are automatically updated for all Applications

DELIMITER $$

DROP TRIGGER IF EXISTS APPLICANT\_EMAIL\_UPDATE $$

CREATE TRIGGER APPLICANT\_EMAIL\_UPDATE AFTER UPDATE ON `Applicant`

FOR EACH ROW

BEGIN

UPDATE `Application` SET ApplicantEmail=NEW.Email WHERE Application.ApplicantID=NEW.ApplicantID ;

END $$

DELIMITER ;

# Create SQL scripts to populate all tables with data

## Populate Country relation

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

-- Country

SELECT "Loading populate\_country.sql" ;

INSERT INTO Country (CountryISOCode, Name)

VALUES

('AD', 'Andorra'),

('AE', 'United Arab Emirates'),

('AF', 'Afghanistan'),

('AG', 'Antigua and Barbuda'),

('AI', 'Anguilla'),

('AL', 'Albania'),

('AM', 'Armenia'),

('AO', 'Angola'),

('AQ', 'Antarctica'),

('AR', 'Argentina'),

('AS', 'American Samoa'),

('AT', 'Austria'),

('AU', 'Australia'),

('AW', 'Aruba'),

('AX', 'Åland Islands'),

('AZ', 'Azerbaijan'),

('BA', 'Bosnia and Herzegovina'),

('BB', 'Barbados'),

('BD', 'Bangladesh'),

('BE', 'Belgium'),

('BF', 'Burkina Faso'),

('BG', 'Bulgaria'),

('BH', 'Bahrain'),

('BI', 'Burundi'),

('BJ', 'Benin'),

('BL', 'Saint Barthélemy'),

('BM', 'Bermuda'),

('BN', 'Brunei Darussalam'),

('BO', 'Bolivia, Plurinational State of'),

('BQ', 'Bonaire, Sint Eustatius and Saba'),

('BR', 'Brazil'),

('BS', 'Bahamas'),

('BT', 'Bhutan'),

('BV', 'Bouvet Island'),

('BW', 'Botswana'),

('BY', 'Belarus'),

('BZ', 'Belize'),

('CA', 'Canada'),

('CC', 'Cocos (Keeling) Islands'),

('CD', 'Congo, the Democratic Republic of the'),

('CF', 'Central African Republic'),

('CG', 'Congo'),

('CH', 'Switzerland'),

('CI', 'Côte d''Ivoire'),

('CK', 'Cook Islands'),

('CL', 'Chile'),

('CM', 'Cameroon'),

('CN', 'China'),

('CO', 'Colombia'),

('CR', 'Costa Rica'),

('CU', 'Cuba'),

('CV', 'Cabo Verde'),

('CW', 'Curaçao'),

('CX', 'Christmas Island'),

('CY', 'Cyprus'),

('CZ', 'Czech Republic'),

('DE', 'Germany'),

('DJ', 'Djibouti'),

('DK', 'Denmark'),

('DM', 'Dominica'),

('DO', 'Dominican Republic'),

('DZ', 'Algeria'),

('EC', 'Ecuador'),

('EE', 'Estonia'),

('EG', 'Egypt'),

('EH', 'Western Sahara'),

('ER', 'Eritrea'),

('ES', 'Spain'),

('ET', 'Ethiopia'),

('FI', 'Finland'),

('FJ', 'Fiji'),

('FK', 'Falkland Islands (Malvinas)'),

('FM', 'Micronesia, Federated States of'),

('FO', 'Faroe Islands'),

('FR', 'France'),

('GA', 'Gabon'),

('GB', 'United Kingdom'),

('GD', 'Grenada'),

('GE', 'Georgia'),

('GF', 'French Guiana'),

('GG', 'Guernsey'),

('GH', 'Ghana'),

('GI', 'Gibraltar'),

('GL', 'Greenland'),

('GM', 'Gambia'),

('GN', 'Guinea'),

('GP', 'Guadeloupe'),

('GQ', 'Equatorial Guinea'),

('GR', 'Greece'),

('GS', 'South Georgia and the South Sandwich Islands'),

('GT', 'Guatemala'),

('GU', 'Guam'),

('GW', 'Guinea-Bissau'),

('GY', 'Guyana'),

('HK', 'Hong Kong'),

('HM', 'Heard Island and McDonald Islands'),

('HN', 'Honduras'),

('HR', 'Croatia'),

('HT', 'Haiti'),

('HU', 'Hungary'),

('ID', 'Indonesia'),

('IE', 'Ireland'),

('IL', 'Israel'),

('IM', 'Isle of Man'),

('IN', 'India'),

('IO', 'British Indian Ocean Territory'),

('IQ', 'Iraq'),

('IR', 'Iran, Islamic Republic of'),

('IS', 'Iceland'),

('IT', 'Italy'),

('JE', 'Jersey'),

('JM', 'Jamaica'),

('JO', 'Jordan'),

('JP', 'Japan'),

('KE', 'Kenya'),

('KG', 'Kyrgyzstan'),

('KH', 'Cambodia'),

('KI', 'Kiribati'),

('KM', 'Comoros'),

('KN', 'Saint Kitts and Nevis'),

('KP', 'Korea, Democratic People''s Republic of'),

('KR', 'Korea, Republic of'),

('KW', 'Kuwait'),

('KY', 'Cayman Islands'),

('KZ', 'Kazakhstan'),

('LA', 'Lao People''s Democratic Republic'),

('LB', 'Lebanon'),

('LC', 'Saint Lucia'),

('LI', 'Liechtenstein'),

('LK', 'Sri Lanka'),

('LR', 'Liberia'),

('LS', 'Lesotho'),

('LT', 'Lithuania'),

('LU', 'Luxembourg'),

('LV', 'Latvia'),

('LY', 'Libya'),

('MA', 'Morocco'),

('MC', 'Monaco'),

('MD', 'Moldova, Republic of'),

('ME', 'Montenegro'),

('MF', 'Saint Martin (French part)'),

('MG', 'Madagascar'),

('MH', 'Marshall Islands'),

('MK', 'Macedonia, the former Yugoslav Republic of'),

('ML', 'Mali'),

('MM', 'Myanmar'),

('MN', 'Mongolia'),

('MO', 'Macao'),

('MP', 'Northern Mariana Islands'),

('MQ', 'Martinique'),

('MR', 'Mauritania'),

('MS', 'Montserrat'),

('MT', 'Malta'),

('MU', 'Mauritius'),

('MV', 'Maldives'),

('MW', 'Malawi'),

('MX', 'Mexico'),

('MY', 'Malaysia'),

('MZ', 'Mozambique'),

('NA', 'Namibia'),

('NC', 'New Caledonia'),

('NE', 'Niger'),

('NF', 'Norfolk Island'),

('NG', 'Nigeria'),

('NI', 'Nicaragua'),

('NL', 'Netherlands'),

('NO', 'Norway'),

('NP', 'Nepal'),

('NR', 'Nauru'),

('NU', 'Niue'),

('NZ', 'New Zealand'),

('OM', 'Oman'),

('PA', 'Panama'),

('PE', 'Peru'),

('PF', 'French Polynesia'),

('PG', 'Papua New Guinea'),

('PH', 'Philippines'),

('PK', 'Pakistan'),

('PL', 'Poland'),

('PM', 'Saint Pierre and Miquelon'),

('PN', 'Pitcairn'),

('PR', 'Puerto Rico'),

('PS', 'Palestine, State of'),

('PT', 'Portugal'),

('PW', 'Palau'),

('PY', 'Paraguay'),

('QA', 'Qatar'),

('RE', 'Réunion'),

('RO', 'Romania'),

('RS', 'Serbia'),

('RU', 'Russian Federation'),

('RW', 'Rwanda'),

('SA', 'Saudi Arabia'),

('SB', 'Solomon Islands'),

('SC', 'Seychelles'),

('SD', 'Sudan'),

('SE', 'Sweden'),

('SG', 'Singapore'),

('SH', 'Saint Helena, Ascension and Tristan da Cunha'),

('SI', 'Slovenia'),

('SJ', 'Svalbard and Jan Mayen'),

('SK', 'Slovakia'),

('SL', 'Sierra Leone'),

('SM', 'San Marino'),

('SN', 'Senegal'),

('SO', 'Somalia'),

('SR', 'Suriname'),

('SS', 'South Sudan'),

('ST', 'Sao Tome and Principe'),

('SV', 'El Salvador'),

('SX', 'Sint Maarten (Dutch part)'),

('SY', 'Syrian Arab Republic'),

('SZ', 'Swaziland'),

('TC', 'Turks and Caicos Islands'),

('TD', 'Chad'),

('TF', 'French Southern Territories'),

('TG', 'Togo'),

('TH', 'Thailand'),

('TJ', 'Tajikistan'),

('TK', 'Tokelau'),

('TL', 'Timor-Leste'),

('TM', 'Turkmenistan'),

('TN', 'Tunisia'),

('TO', 'Tonga'),

('TR', 'Turkey'),

('TT', 'Trinidad and Tobago'),

('TV', 'Tuvalu'),

('TW', 'Taiwan, Province of China'),

('TZ', 'Tanzania, United Republic of'),

('UA', 'Ukraine'),

('UG', 'Uganda'),

('UM', 'United States Minor Outlying Islands'),

('US', 'United States'),

('UY', 'Uruguay'),

('UZ', 'Uzbekistan'),

('VA', 'Holy See (Vatican City State)'),

('VC', 'Saint Vincent and the Grenadines'),

('VE', 'Venezuela, Bolivarian Republic of'),

('VG', 'Virgin Islands, British'),

('VI', 'Virgin Islands, U.S.'),

('VN', 'Viet Nam'),

('VU', 'Vanuatu'),

('WF', 'Wallis and Futuna'),

('WS', 'Samoa'),

('YE', 'Yemen'),

('YT', 'Mayotte'),

('ZA', 'South Africa'),

('ZM', 'Zambia'),

('ZW', 'Zimbabwe');

## Populate lookup relations

SELECT "Loading populate\_lookups.sql" ;

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

-- ApplicationStatus

-- [10000..10499] reserved for all kinds of ongoing statuses

-- [10500..10999] reserved for all kinds of completed statuses

INSERT INTO `Application Status` (ApplicationStatusID, Status, Description)

VALUES (10000, 'ongoing',

'Application/information gathering is currently ongoing' ) ;

INSERT INTO `Application Status` (ApplicationStatusID, Status, Description)

VALUES (10500, 'complete.accepted',

'Application accepted. Elevated to RHD office.' ) ;

INSERT INTO `Application Status` (ApplicationStatusID, Status, Description)

VALUES (10501, 'complete.declined',

'Application declined. School chooses not to pursue.' ) ;

INSERT INTO `Application Status` (ApplicationStatusID, Status, Description)

VALUES (10502, 'complete.withdrawn', 'Application withdrawn by applicant.' ) ;

INSERT INTO `Application Status` (ApplicationStatusID, Status, Description)

VALUES (10503, 'complete.lapsed', 'No activity for a significant period.' ) ;

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

-- DocumentType

INSERT INTO `Document Type` (DocTypeID, Type, Description)

VALUES (20000, 'application', 'A completed RHD application form.') ;

INSERT INTO `Document Type` (DocTypeID, Type, Description)

VALUES (20001, 'cv', 'CV') ;

INSERT INTO `Document Type` (DocTypeID, Type, Description)

VALUES (20002, 'resume', 'resume') ;

INSERT INTO `Document Type` (DocTypeID, Type, Description)

VALUES (20003, 'faculty\_assessment\_memo', 'Faculty assessment memo') ;

INSERT INTO `Document Type` (DocTypeID, Type, Description)

VALUES (20004, 'certificate', 'A certificate of a previous degree.') ;

INSERT INTO `Document Type` (DocTypeID, Type, Description)

VALUES (20005, 'transcript', 'An academic transcript of a previous degree.') ;

INSERT INTO `Document Type` (DocTypeID, Type, Description)

VALUES (20006, 'thesis', 'An previous degree major work.') ;

INSERT INTO `Document Type` (DocTypeID, Type, Description)

VALUES (20007, 'proposal', 'An application proposal.') ;

INSERT INTO `Document Type` (DocTypeID, Type, Description)

VALUES (20008, 'reference',

'A character/academic reference of the applicant.') ;

INSERT INTO `Document Type` (DocTypeID, Type, Description)

VALUES (20009, 'publication',

'A scientific publication authored by the applicant.') ;

INSERT INTO `Document Type` (DocTypeID, Type, Description)

VALUES (20010, 'financial',

'A document relating to how the RHD place is to be funded.') ;

INSERT INTO `Document Type` (DocTypeID, Type, Description)

VALUES (20011, 'general', 'A type of document other than decribed above.') ;

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

-- DocumentStatus

INSERT INTO `Document Status` (DocStatusID, Status, Description)

VALUES (30000, 'original.english', 'Original document in English');

INSERT INTO `Document Status` (DocStatusID, Status, Description)

VALUES (30001, 'original.lote', 'Original document in a language other than English.');

INSERT INTO `Document Status` (DocStatusID, Status, Description)

VALUES (30002, 'translation', 'Official translation into English of original document');

INSERT INTO `Document Status` (DocStatusID, Status, Description)

VALUES (30003, 'summary', 'Not an original source document.');

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

-- AwardType

INSERT INTO `Award Type` (AwardID, Type, Description)

VALUES (40000, 'PhD', 'PhD in any field in the school');

INSERT INTO `Award Type` (AwardID, Type, Description)

VALUES (40001, 'PhD (Comp Sc)', 'PhD in Computer Science');

INSERT INTO `Award Type` (AwardID, Type, Description)

VALUES (40002, 'masters', 'Masters by research in any field in the school');

INSERT INTO `Award Type` (AwardID, Type, Description)

VALUES (40003, 'MIT', 'Masters IT');

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

-- ResearchArea

INSERT INTO `Research Area` (FORCode, Description, ResearchArea, GeneralArea)

VALUES (100503, 'computer communication networks', '', '');

INSERT INTO `Research Area` (FORCode, Description, ResearchArea, GeneralArea)

VALUES (080199, 'artificial intelligence and image processing not elsewhere classified', '', '');

INSERT INTO `Research Area` (FORCode, Description, ResearchArea, GeneralArea)

VALUES (080399, 'computer software not elsewhere classified', '', '');

INSERT INTO `Research Area` (FORCode, Description, ResearchArea, GeneralArea)

VALUES (080499, 'data format not elsewhere classified', '', '');

INSERT INTO `Research Area` (FORCode, Description, ResearchArea, GeneralArea)

VALUES (080699, 'information systems not elsewhere classified', '', '');

INSERT INTO `Research Area` (FORCode, Description, ResearchArea, GeneralArea)

VALUES (089999, 'information and computing sciences not elsewhere classified', '', '');

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

-- DecisionTypes

INSERT INTO `Decision Type` (DecisionTypeID, type)

VALUES (50000, 'GPA too low');

INSERT INTO `Decision Type` (DecisionTypeID, type)

VALUES (50001, 'TOEFL/IELTS score too low');

INSERT INTO `Decision Type` (DecisionTypeID, type)

VALUES (50002, 'Hand over');

INSERT INTO `Decision Type` (DecisionTypeID, type)

VALUES (50003, 'RFI');

INSERT INTO `Decision Type` (DecisionTypeID, type)

VALUES (50500, 'change.research\_area.addition');

INSERT INTO `Decision Type` (DecisionTypeID, type)

VALUES (50501, 'change.research\_area.deletion');

INSERT INTO `Decision Type` (DecisionTypeID, type)

VALUES (50502, 'change.research\_area\_oversees.addition');

INSERT INTO `Decision Type` (DecisionTypeID, type)

VALUES (50503, 'change.research\_area\_oversees.deletion');

INSERT INTO `Decision Type` (DecisionTypeID, type)

VALUES (50504, 'change.supervisor.addition');

INSERT INTO `Decision Type` (DecisionTypeID, type)

VALUES (50505, 'change.supervisor.deletion');

INSERT INTO `Decision Type` (DecisionTypeID, type)

VALUES (50506, 'change.flag.addition');

INSERT INTO `Decision Type` (DecisionTypeID, type)

VALUES (50507, 'change.flag.deletion');

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

-- CorrespondenceMethod

INSERT INTO `Correspondence Method` (CorrMethodID, Method)

VALUES (60000, 'email');

INSERT INTO `Correspondence Method` (CorrMethodID, Method)

VALUES (60001, 'telephone');

INSERT INTO `Correspondence Method` (CorrMethodID, Method)

VALUES (60002, 'carrier pigeon');

INSERT INTO `Correspondence Method` (CorrMethodID, Method)

VALUES (69999, 'other');

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

-- Payment Method

INSERT INTO `Payment Method` (PayMethodID, Method)

VALUES (70000, 'Savings'), (70001, 'scholarship'), (70002, 'HECS-HELP'),

(70003, 'Research Training Scheme (RTS)');

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

-- Visa Status

INSERT INTO `Visa Status` (VisaStatusID, Status, description)

VALUES (80000,'not required', 'the applicant is an australian/NZ resident and does not require a visa'),

(80001,'unknown', 'the students visa status is not currently known'),

(80002, 'approved-nodoc', 'Stated the visa has been approved but has not supplied a document to that effect'),

(80003, 'approved-doc', 'The visa has been approved and has supplied a document that proves it'),

(80004, 'submitted', 'an application has been submitted and is currently being processed');

## Populate staff information

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

-- UNIVERSITY STAFF MEMBERS

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

SELECT "Loading populate\_staff.sql" ;

-- Denise de Vries

INSERT INTO `University Staff Member` (StaffID, LName, FName, canSupervise,

email)

VALUES (1000, 'de Vries', 'Denise', 1, 'denise.deries@flinders.edu.au');

-- FOR 'computer software not elsewhere classified'

INSERT INTO `University Staff Member\_Research Area` (StaffID, FORCode)

VALUES (1000, 080399);

-- FOR 'data format not elsewhere classified'

INSERT INTO `University Staff Member\_Research Area` (StaffID, FORCode)

VALUES (1000, 080499);

-- FOR 'information systems not elsewhere classified'

INSERT INTO `University Staff Member\_Research Area` (StaffID, FORCode)

VALUES (1000, 080699);

-- FOR 'information and computing sciences not elsewhere classified'

INSERT INTO `University Staff Member\_Research Area` (StaffID, FORCode)

VALUES (1000, 089999);

-- Denise oversees data format

INSERT INTO `University Staff Member\_Research Area2` (StaffID, FORCode)

VALUES (1000, 080499);

-- Denise oversees info systems

INSERT INTO `University Staff Member\_Research Area2` (StaffID, FORCode)

VALUES (1000, 89999);

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

-- Paul Calder

INSERT INTO `University Staff Member` (StaffID, LName, FName, canSupervise,

email)

VALUES (1001, 'Calder', 'Paul', 1, 'paul.calder@flinders.edu.au');

-- FOR 'artificial intelligence and image processing not elsewhere classified'

INSERT INTO `University Staff Member\_Research Area` (StaffID, FORCode)

VALUES (1001, 080199);

-- FOR 'computer software not elsewhere classified'

INSERT INTO `University Staff Member\_Research Area` (StaffID, FORCode)

VALUES (1001, 080399);

-- FOR 'computer communication networks'

INSERT INTO `University Staff Member\_Research Area` (StaffID, FORCode)

VALUES (1001, 100503);

-- Paul oversees computer software

INSERT INTO `University Staff Member\_Research Area2` (StaffID, FORCode)

VALUES (1001, 080399);

-- Paul oversees 'computer communication networks'

INSERT INTO `University Staff Member\_Research Area2` (StaffID, FORCode)

VALUES (1001, 100503);

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

-- John Roddick

INSERT INTO `University Staff Member` (StaffID, LName, FName, canSupervise,

email)

VALUES (1002, 'Roddick', 'John', 1, 'john.roddick@flinders.edu.au');

-- FOR 'artificial intelligence and image processing not elsewhere classified'

INSERT INTO `University Staff Member\_Research Area` (StaffID, FORCode)

VALUES (1002, 080199);

-- FOR 'information systems not elsewhere classified'

INSERT INTO `University Staff Member\_Research Area` (StaffID, FORCode)

VALUES (1002, 080699);

-- John Roddick oversees AI

INSERT INTO `University Staff Member\_Research Area2` (StaffID, FORCode)

VALUES (1001, 80199);

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

-- Jennie Brand

INSERT INTO `University Staff Member` (StaffID, LName, FName, canSupervise,

email)

VALUES (1003, 'Brand', 'Jennie', 0, 'jennie.brand@flinders.edu.au');

## Populate applicant and applicant information

SELECT "Loading populate\_apps.sql" ;

-- IDs

-- Applicant 1-9-

-- Application x11 - x19

-- Degree x21 - x29

-- Document x31 - x39

-- Correspondence x41 - x49

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

-- Reset tables and import other info

DELETE FROM `Application\_Research Area`;

DELETE FROM Decision;

DELETE FROM Degree;

DELETE FROM Document;

DELETE FROM Application;

DELETE FROM Applicant;

DELETE FROM Correspondence;

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

-- Application for PhD studies.txt

-- 01

SELECT "Populating Applicant 1 info" ;

INSERT INTO Applicant (ApplicantID, FName, LName, Email, Mobile, StreetAddress,

City, Postcode, AddressCountryISOCode, DateAdded, LastModifiedByStaffID)

VALUES (01, 'Shirin', 'Ebadi', 'shirin.ebadi@keb.com.de',

'+49 (176) 6488 9999', 'Zwillingstr 99, App 099', 'Munich', 80807, 'DE',

'2014-05-01', 1001);

INSERT INTO Degree (DegID, ApplicantID, Name, Type, YearCompleted,

InstitutionName, InstitutionCountryISOCode)

VALUES (121, 01, 'Electrical Engineering with expertise in Control

Theory and Reat-Time Applications', 'bachelor', '2005-12-31',

'University of Tabriz', 'IR') ;

INSERT INTO Degree (DegID, ApplicantID, Name, Type, YearCompleted,

InstitutionName, InstitutionCountryISOCode)

VALUES (122, 01, 'Electrical Engineering with expertise in Control

Theory and Reat-Time Applications', 'masters', '2008-12-31',

'University of Tabriz', 'IR') ;

INSERT INTO Application (ApplicationID, ApplicantID, awardID, DateAdded,

DateLastChecked, DateLastModified, LastModifiedByStaffID, applicationStatusID)

VALUES (111, 01, 40000, '2014-05-01', '2014-05-01', '2014-05-01', 1002, 10000) ;

INSERT INTO Document (DocID, UploadLink, ApplicantID, DocTypeID, DocStatusID)

VALUES (131, '/mnt/data/rhd/01/CV/001.pdf', 01, 20001, 30000);

-- Denise to supervise this

INSERT INTO `Supervise as` (PrimarySupervisor, ApplicationID, StaffID)

VALUES (1, 111, 1000);

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

-- FW Your kind supervision for my intended PhD.txt

-- 2

SELECT "Populating Applicant 2 info" ;

INSERT INTO Applicant (ApplicantID, FName, LName, Email, Mobile, Phone,

DateAdded, LastModifiedByStaffID)

VALUES (2, 'Mohammad', 'Almalki', 'don.memo@hotmail.com', '+966 565907070',

'+966 12 527000000 ext 4951', '2014-05-02', 1003);

INSERT INTO Application (ApplicantID, ApplicationID, awardID, DateAdded,

DateLastChecked, DateLastModified, LastModifiedByStaffID, applicationStatusID)

VALUES (2, 211, 40000, '2014-05-02', '2014-05-02', '2014-05-02', 1003, 10000) ;

INSERT INTO Degree (DegID, ApplicantID, Name, Type, YearCompleted,

InstitutionName, InstitutionCountryISOCode)

VALUES (221, 2, 'IT', 'master', '2010-12-31',

'University of Technology Sydney', 'AU') ;

INSERT INTO Document (UploadLink, ApplicantID, DocTypeID, DocStatusID)

VALUES ('/mnt/data/rhd/02/CV/001.pdf',

02, 20001, 30000) ;

INSERT INTO Document (UploadLink, ApplicantID, DocTypeID, Description,

DocStatusID)

VALUES ('/mnt/data/rhd/02/transcript/001.pdf',

02, 20005, 'transcript of masters course', 30000) ;

INSERT INTO Document (UploadLink, ApplicantID, DocTypeID, Description,

DocStatusID)

VALUES ('/mnt/data/rhd/02/certificate/001.pdf',

02, 20004, 'bachelors certificate', 30000) ;

INSERT INTO Document (UploadLink, ApplicantID, ApplicationID, DocTypeID,

DocStatusID)

VALUES ('/mnt/data/rhd/02/proposal/001.pdf',

02, 211, 20007, 30000) ;

-- Denise to supervise this (associate supervision)

INSERT INTO `Supervise as` (PrimarySupervisor, ApplicationID, StaffID)

VALUES (0, 211, 1000);

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

-- Fwd Flinders Application - PhD (Comp Sc) - Sem 2 2015.txt

-- 3

SELECT "Populating Applicant 3 info" ;

INSERT INTO Applicant (ApplicantID, FName, LName, StudentID, DateAdded,

LastModifiedByStaffID)

VALUES (03, 'Azzam', 'Alwash', '1234567', '2014-05-03', 1003) ;

INSERT INTO Application (ApplicationID, ApplicantID, awardID, ProposalSummary,

ProposedStartDate, DateAdded, DateLastChecked, DateLastModified,

LastModifiedByStaffID, applicationStatusID)

VALUES (311, 03, 40001,

'Genetic algorithms for Arabic character recognition', '2015-07-01',

'2014-05-03', '2014-05-03', '2014-05-03', 1001, 10000) ;

INSERT INTO Degree (ApplicantID, Name, Type, GPA, InstitutionCountryISOCode)

VALUES (03, 'IS and CS', 'bachelors', 4.27, 'IQ') ;

INSERT INTO Degree (ApplicantID, Name, Type, GPA, InstitutionCountryISOCode)

VALUES (03, 'IT', 'masters', 6.79, 'MY') ;

INSERT INTO Document (UploadLink, ApplicantID, ApplicationID, DocTypeID,

DocStatusID)

VALUES ('/mnt/data/rhd/03/Application/0001.pdf', 03, 311, 20000, 30000);

-- Denise to supervise this

INSERT INTO `Supervise as` (PrimarySupervisor, ApplicationID, StaffID)

VALUES (1, 311, 1000);

-- Paul Calder to supervise this (associate)

INSERT INTO `Supervise as` (PrimarySupervisor, ApplicationID, StaffID)

VALUES (0, 311, 1001);

-- John Roddick to supervise this (associate)

INSERT INTO `Supervise as` (PrimarySupervisor, ApplicationID, StaffID)

VALUES (0, 311, 1002);

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

-- Fwd Flinders Application - PhD (Computer Science) Sem 2 2014 .txt

-- 4

SELECT "Populating Applicant 4 info" ;

INSERT INTO Applicant (ApplicantID, FName, LName, StudentID, DateAdded,

LastModifiedByStaffID)

VALUES (04, 'Mustafa', 'Al Lami', 2130106, '2014-05-04', 1000);

INSERT INTO Application (ApplicationID, ApplicantID, awardID, ProposalSummary,

DateAdded, DateLastChecked, DateLastModified, LastModifiedByStaffID,

applicationStatusID)

VALUES (411, 04, 40000, 'Cloud computing for large scale data analysis',

'2014-05-04', '2014-05-04', '2014-05-04', 1003, 10000) ;

INSERT INTO Document (UploadLink, ApplicantID, DocTypeID, DocStatusID)

VALUES ('/mnt/data/rhd/04/faculty\_assessment\_memo/0001.pdf',

04, 20003, 30000);

INSERT INTO Document (UploadLink, ApplicantID, DocTypeID, DocStatusID)

VALUES ('/mnt/data/rhd/04/general/0001.pdf',

04, 20011, 30000);

INSERT INTO Document (UploadLink, ApplicantID, DocTypeID, DocStatusID)

VALUES ('/mnt/data/rhd/04/general/0002.pdf', 04, 20011, 30000);

-- Paul Colder asks for more info about masters

INSERT INTO Decision ( DecID, ApplicationID, Date, DecisionTypeID, Comment,

StaffID)

VALUES (1, 411, '2014.05.15', 50003,

'Have asked for more info about Masters project', 1001 );

INSERT INTO Correspondence (CorrID, `Date`, CorrMethodID, Summary,

ApplicationID, StaffID)

VALUES (441, '2014-05-04', 60000, 'Initial inquiry from applicant', 411, 1001);

INSERT INTO Correspondence (CorrID, `Date`, CorrMethodID, Summary,

ApplicationID, StaffID)

VALUES (442, '2014-05-04', 60000, 'Asked applicant for more information', 411,

1001);

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

-- Fwd Flinders Application.txt

-- 5

SELECT "Populating Applicant 5 info" ;

INSERT INTO Applicant (ApplicantID, FName, StudentID, DateAdded,

LastModifiedByStaffID)

VALUES (05, 'Ena', '2345678', '2014-05-05', 1001) ;

INSERT INTO Application (ApplicationID, ApplicantID, ProposedStartDate,

DateAdded, DateLastChecked, DateLastModified, LastModifiedByStaffID,

applicationStatusID)

VALUES (511, 05, '2014-07-01', '2014-05-05', '2014-05-05', '2014-05-05', 1001,

10000) ;

INSERT INTO Document (UploadLink, ApplicantID, ApplicationID, DocTypeID,

DocStatusID)

VALUES ('/mnt/data/rhd/05/proposal/0001.pdf', 05, 511, 20007, 30000);

INSERT INTO Document (UploadLink, ApplicantID, ApplicationID, DocTypeID,

DocStatusID)

VALUES ('/mnt/data/rhd/05/Application/0001.pdf', 05, 511, 20000, 30000);

INSERT INTO Document (UploadLink, ApplicantID, DocTypeID, Description,

DocStatusID)

VALUES ('/mnt/data/rhd/05/transcript/0001.pdf', 05, 20005,

'bachelor certificate and transcript', 30000);

INSERT INTO Document (UploadLink, ApplicantID, DocTypeID, Description,

DocStatusID)

VALUES ('/mnt/data/rhd/05/transcript/0002.pdf', 05, 20005,

'master certificate and transcript', 30000);

INSERT INTO Document (UploadLink, ApplicantID, DocTypeID, DocStatusID)

VALUES ('/mnt/data/rhd/05/financial/0001.pdf',

05, 20010, 30000);

INSERT INTO Document (UploadLink, ApplicantID, ApplicationID, DocTypeID,

Description, DocStatusID)

VALUES ('/mnt/data/rhd/05/reference/0001.pdf',

05, 511, 20008,

'recommendation letters', 30000);

INSERT INTO Document (UploadLink, ApplicantID, ApplicationID, DocTypeID,

Description, DocStatusID)

VALUES ('/mnt/data/rhd/05/general/0001.pdf',

05, 511, 20011, 'training certificate', 30000);

INSERT INTO Document (UploadLink, ApplicantID, DocTypeID, DocStatusID)

VALUES ('/mnt/data/rhd/05/faculty\_assessment\_memo/0001.pdf', 05,

20003, 30000);

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

-- Fwd PhD inquiry.txt

-- 6

SELECT "Populating Applicant 6 info" ;

INSERT INTO Applicant (ApplicantID, FName, LName, Sex, DateAdded,

LastModifiedByStaffID)

VALUES (06, 'Fakhri', 'Bazzaz', 1, '2014-05-05', 1002) ;

INSERT INTO Application (ApplicationID, ApplicantID, awardID, DateAdded,

DateLastChecked, DateLastModified, LastModifiedByStaffID, applicationStatusID)

VALUES (611, 06, 40000, '2014-05-06', '2014-05-06', '2014-05-06', 1002, 10000);

-- 100503 Computer Communications Networks

INSERT INTO `Application\_Research Area` (ApplicationID, FORCode)

VALUES (611, 100503) ;

INSERT INTO Document (UploadLink, ApplicantID, DocTypeID, Description,

DocStatusID)

VALUES ('/mnt/data/rhd/06/general/0001.pdf', 06, 20011, 'Award of Degree',

30000);

INSERT INTO Document (UploadLink, ApplicantID, ApplicationID, DocTypeID,

DocStatusID)

VALUES ('/mnt/data/rhd/06/certificate/0001.pdf', 06, 611, 20004, 30000);

INSERT INTO Document (UploadLink, ApplicantID, DocTypeID, DocStatusID)

VALUES ('/mnt/data/rhd/06/cv/0001.pdf', 06, 20001, 30000) ;

INSERT INTO Document (UploadLink, ApplicantID, DocTypeID, Description,

DocStatusID)

VALUES ('/mnt/data/rhd/06/certificate/0002.pdf', 06, 20004, 'English cert',

30000);

-- add the thesis to documents

INSERT INTO Document (UploadLink, ApplicantID, DocTypeID, Description,

DocStatusID)

VALUES ('/mnt/data/rhd/06/publication/0001.pdf', 06, 20006, 'masters thesis',

30000);

-- add the transcript to documents

INSERT INTO Document (UploadLink, ApplicantID, DocTypeID, Description,

DocStatusID)

VALUES ('/mnt/data/rhd/06/transcript/0001.pdf', 06, 20005, 'masters transcript',

30000);

-- add a reference statement as a document

INSERT INTO Document (UploadLink, ApplicantID, ApplicationID, DocTypeID,

DocStatusID)

VALUES ('/mnt/data/rhd/06/reference/0001.pdf', 06, 611, 20008, 30000) ;

-- add the other reference statement as a document

INSERT INTO Document (UploadLink, ApplicantID, ApplicationID, DocTypeID,

DocStatusID)

VALUES ('/mnt/data/rhd/06/reference/0002.pdf', 06, 611, 20008, 30000) ;

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

-- Fwd Requesting for PhD supervision.txt

-- 7

SELECT "Populating Applicant 7 info" ;

INSERT INTO Applicant (ApplicantID, FName, DateAdded, Email,

LastModifiedByStaffID)

VALUES (07, 'Nemo', '2014-05-07', 'nemo@gmail.com', 1000) ;

INSERT INTO Degree (DegID, ApplicantID, Name, Type, InstitutionName,

InstitutionCountryISOCode)

VALUES (721, 07, 'IT', 'bachelors', 'Manipal University', 'IN') ;

INSERT INTO Degree (DegID, ApplicantID, Name, Type, InstitutionName,

InstitutionCountryISOCode)

VALUES (722, 07, 'Software development and engineering',

'masters', 'West Bengal University', 'IN') ;

INSERT INTO Application (ApplicationID, ApplicantID, awardID, DateAdded,

DateLastChecked, DateLastModified, LastModifiedByStaffID, applicationStatusID)

VALUE (711, 07, 40000, '2014-05-07', '2014-05-07', '2014-05-07', 1000, 10000) ;

INSERT INTO Document (UploadLink, ApplicantID, DocTypeID, DocStatusID)

VALUES ('/mnt/data/rhd/07/resume/0001.pdf', 07, 20002, 30000);

-- add the publication to documents

INSERT INTO Document (UploadLink, ApplicantID, DocTypeID, Description,

DocStatusID)

VALUES ('/mnt/data/rhd/07/publication/0001.pptx', 07, 20009, 'Six-plus

presenting a six-element framework for sentiment analysis.pptx', 30000);

INSERT INTO Document (UploadLink, ApplicantID, DocTypeID, Description,

DocStatusID)

VALUES ('/mnt/data/rhd/07/publication/0002.docx', 07, 20009, 'The price

prediction framework based upon representation of the opinions', 30000);

INSERT INTO Document (UploadLink, ApplicantID, DocTypeID, Description,

DocStatusID)

VALUES ('/mnt/data/rhd/07/publication/0003.docx', 07, 20009,

'The classification framework for the representation of opinions', 30000);

-- Denise de Vries flags interest in this application

INSERT INTO `University Staff Member\_Application` (StaffID, ApplicationID)

VALUES (1000, 711);

-- John Roddick flags interest in this application

INSERT INTO `University Staff Member\_Application` (StaffID, ApplicationID)

VALUES (1002, 711);

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

-- PhD Student.txt

-- 8

SELECT "Populating Applicant 8 info" ;

INSERT INTO Applicant (ApplicantID, FName, Email, DateAdded,

LastModifiedByStaffID)

VALUES (08, 'Sara', 'sara@gmail.com', '2014-05-08', 1003) ;

INSERT INTO Degree (DegID, ApplicantID, Name, Type, YearCompleted, GPA,

InstitutionCountryISOCode)

VALUES (821, 08, 'Masters Comp Sc (Information Security)', 'masters',

'2013-12-31', 6.79, 'MY');

INSERT INTO Application (ApplicationID, ApplicantID, awardID, DateAdded,

DateLastChecked, DateLastModified, LastModifiedByStaffID, applicationStatusID)

VALUES (811, 08, 40000, '2014-05-08', '2014-05-08', '2014-05-08', 1003, 10000);

-- some research areas

INSERT INTO `Application\_Research Area` (ApplicationID, FORCode)

VALUES (811, 100503) ;

INSERT INTO `Application\_Research Area` (ApplicationID, FORCode)

VALUES (811, 89999) ;

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

-- PhD Student1.txt

-- 9

SELECT "Populating Applicant 9 info" ;

INSERT INTO Applicant (ApplicantID, FName, LName, Email, DateAdded,

LastModifiedByStaffID)

VALUES (09, 'Abdul-Allah', 'Al-Sadhan', 'abdul-allah.al-sadhan@gmail.com',

'2014-05-09', 1000) ;

INSERT INTO Application (ApplicationID, ApplicantID, awardID, DateAdded,

DateLastChecked, DateLastModified, LastModifiedByStaffID, applicationStatusID)

VALUES (911, 09, 40000, '2014-05-09', '2014-05-09', '2014-05-09', 1000, 10000);

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

-- PhD Student2.txt

-- 10

SELECT "Populating Applicant 10 info" ;

INSERT INTO Applicant (ApplicantID, FName, LName, Email, DateAdded,

LastModifiedByStaffID)

VALUES (10, 'Fahd', 'Al-Hayyan', 'fahd.al-hayyan@ut.edu.sa', '2014-05-10', 1002)

;

INSERT INTO Application (ApplicationID, ApplicantID, awardID, DateAdded,

DateLastChecked, DateLastModified, LastModifiedByStaffID, applicationStatusID)

VALUES (1011, 10, 40000, '2014-05-10', '2014-05-10', '2014-05-10', 1002, 10000);

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

-- Request for PhD Supervision.txt

-- 11

SELECT "Populating Applicant 11 info" ;

INSERT INTO Applicant (ApplicantID, FName, LName, Email, DateAdded,

LastModifiedByStaffID)

VALUES (11, 'Venkatraman', 'Ramakrishnan',

'venkatraman.ramakrishnan@gmail.com', '2014-05-11', 1003);

INSERT INTO Application (ApplicationID, ApplicantID, awardID, DateAdded,

DateLastChecked, DateLastModified, LastModifiedByStaffID, applicationStatusID)

VALUES (1111, 11, 40000, '2014-05-11', '2014-05-11', '2014-05-11', 1003, 10000);

-- Publication (apps)

INSERT INTO `Publication` (PubID, ApplicantID, Title, Abstract, Publication, IssueNo, IssueDate, OnlineLink, OtherAuthorsNames, Language)

VALUES (2000, 09, "AI in the movies", "This paper details AI portral in the media and the questions it raise and how they are answered in the litrature ...","Science",7,'5/5/2012',"www.science.com","ronald hume","english" );

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

-- referee (apps)

INSERT INTO `Referee` (RefID,ApplicationID,Name,Relation,Phone,Email,Profession,AcademicLink,EnglishSpeaker,EnglishLiterate)

VALUES (1111,111,"james brown","teacher","012456546","James@brown.com","singer","http://www.linkedin.com/pub/James-Brown/11/2d3/a40",1,0);

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

-- Visa (apps)

INSERT INTO `Visa` (VisaID, CountryISOCode, ValidFrom, ValidTo, ApplicantID, VisaStatusID)

VALUES (3001,'DE','01/01/2013', '30/12/2015', 01, 80002);

# Create SQL scripts for required queries

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

-- Required queries for user transactions

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

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

-- a)

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

-- a1) List all the attributes of an Applicant, given a name

SELECT \*

FROM Applicant

WHERE Applicant.FName = 'Shirin'

AND Applicant.LName = 'Ebadi' ;

-- a2) List all the attributes of Publication rows associated with an applicant,

-- given an applicant name

SELECT P.\*

FROM Publication P

INNER JOIN (Applicant A) ON P.ApplicantID = A.ApplicantID

WHERE A.FName = 'Abdul-Allah' AND A.LName = 'Al-Sadhan' ;

-- a3) List all attributes of Degree rows associated with an applicant,

-- given an applicant name

SELECT D.\*

FROM Degree D

INNER JOIN Applicant A ON D.ApplicantID = A.ApplicantID

WHERE A.FName = 'Shirin' AND A.LName = 'Ebadi' ;

-- a4) List all attributes of Visa rows associated with an applicant,

-- given an applicant name

SELECT V.\*

FROM Visa V

INNER JOIN Applicant A ON V.ApplicantID = A.ApplicantID

WHERE A.FName = 'Shirin' AND A.LName = 'Ebadi' ;

-- a5) List all attributes of Document rows associated with an applicant,

-- given an applicant name

SELECT D.\*, DT.Type

FROM Document D

INNER JOIN Applicant A ON D.ApplicantID = A.ApplicantID

INNER JOIN `Document Type` DT ON D.DocTypeID = DT.DocTypeID

WHERE A.FName = 'Shirin' AND A.LName = 'Ebadi' ;

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

-- b) Look up applicant's applications by applicant name

-- Applicant\_Expanded is a view that resolves lookup table IDs to values

SELECT AE.\*

FROM Application\_Expanded AE

INNER JOIN Applicant Appt ON Appt.ApplicantID = AE.ApplicantID

WHERE Appt.FName = 'Shirin' AND Appt.LName = 'Ebadi' ;

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

-- c) Look up applicant's applications by applicant email

SELECT AE.\*

FROM Application\_Expanded AE

INNER JOIN Applicant Appt ON Appt.ApplicantID = AE.ApplicantID

WHERE Appt.Email = 'don.memo@hotmail.com' ;

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

-- d) Look up incomplete applications

SELECT AE.\*

FROM Application\_Expanded AE

WHERE AE.applicationStatusID < 10500 ;

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

-- e) Look up all correspondences relevant to an application

SELECT Corr.\*

FROM Application App

INNER JOIN Correspondence Corr ON Corr.ApplicationID = App.ApplicationID

INNER JOIN Applicant Appt ON Appt.ApplicantID = App.ApplicantID

WHERE Appt.FName = 'Mustafa' AND Appt.LName = 'Al Lami' ;

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

-- f) Is demonstrated in test plan populate scripts

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

-- g) Look up which staff member updated an Application most recently. Project

-- the StaffID, their first name and last name.

SELECT USM.StaffID, USM.FName, USM.LName

FROM Application App

INNER JOIN `University Staff Member` USM

ON USM.StaffID = App.LastModifiedByStaffID

WHERE App.ApplicationID = 311 ;

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

-- h) Check for any decision recorded about an application. Project all

-- Decision attributes.

SELECT Decision.\*, DT.type

FROM Application App

INNER JOIN Decision ON Decision.ApplicationID = App.ApplicationID

INNER JOIN `Decision Type` DT

ON DT.DecisionTypeID = Decision.DecisionTypeID

WHERE App.ApplicationID = 411 ;

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

-- k) Look up an existing application and list outstanding information

-- (checklist). Use utility view

SELECT AQ.\*

FROM Application\_Quickview AQ

WHERE AQ.ApplicationID = 611 ;

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

-- l) Update the checklist to confirm that a mandatory information requirement

-- has been met.

UPDATE Application

SET AddressConfirmed = 1

WHERE Application.ApplicationID = 611 ;

-- expect success

-- rerun k), expect AddressConfirmed == 1/true

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

-- m) Retrieve all on-going applications for which the user has made the most

-- recent correspondence

SELECT AE.\*

FROM Application\_Expanded AE

INNER JOIN `Application Status` AppStat

ON AppStat.ApplicationStatusID = AE.ApplicationStatusID

WHERE AppStat.Status LIKE 'ongoing%'

AND AE.LastModifiedByStaffID = CURRENT\_RHD\_USER();

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

-- o) Update the status of an application. Do nested join to retrieve the ID

-- of the new status, given its string representation.

UPDATE Application App

JOIN

(SELECT \* FROM `Application Status` AppStat

WHERE AppStat.Status = 'complete.declined') AS Lookup

SET App.applicationStatusID = Lookup.ApplicationStatusID

WHERE App.ApplicationID = 611 ;

-- expect success

-- rerun k), expect Status == 'complete.declined'

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

-- p) Look up, add to, and delete from own current research areas

-- p1) look up current research areas

SELECT RA.FORCode, RA.Description

FROM `Research Area` RA

INNER JOIN `University Staff Member\_Research Area` USMRA

ON USMRA.FORCode = RA.FORCode

INNER JOIN `University Staff Member` USM ON USMRA.StaffID = USM.StaffID

WHERE USM.StaffID = 1000 ;

-- p2) Insert new research areas tested in populate script

-- p3) Delete research areas

DELETE `University Staff Member\_Research Area` USMRA

FROM `University Staff Member\_Research Area` USMRA

JOIN

(SELECT RA.FORCode FROM `Research Area` RA

WHERE Description LIKE 'information systems not elsewhere classified')

AS Lookup

WHERE StaffID = 1000

AND USMRA.FORCode = Lookup.FORCode;

-- rerun p1) to see a different result

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

-- q) Search for all applications in certain research areas that have been added

-- since a certain time. Use the academic staff view.

SELECT \*

FROM Recent\_Applications\_In\_My\_Research\_Area;

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

-- s) Retrieve all staff who have flagged an application, or have edited an

-- application. Use an RHD staff view.

SELECT \*

FROM Application\_Staff\_Overview;

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

-- t) Retrieve all ongoing applications. Use a utility view.

SELECT \*

FROM Application\_Quickview AQ

WHERE AQ.Status LIKE 'ongoing%';

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

-- u) List all applications waiting for supervisor agreement. Use an RHD staff

-- view. This view is intended to help RHD staff identify a staff member who may

-- can supervise the application. The following columns are projected:

--

-- ApplicationID

-- FName - Applicant's first name

-- LName - Applicant's last name

-- Email - Applicant's email

-- DateAdded - the date the application was first added to the

-- system

-- Associate supervisor count - the number of associate (not primary)

-- supervisors registered for this application

-- FORCode - Field of Research codes identified for the

-- application. There may be multiple of these, so

-- the row may be repeated

-- StaffOverseeingAreaFName - the first name of the member of staff who

-- oversees the FORCode research area

-- StaffOverseeingAreaLName - the last name of the member of staff who

-- oversees the FORCode research area

SELECT \*

FROM Application\_Without\_Supervisor;

# Monitor and tune the operational system

Possible performance enhancements

# Update test plan

# Create SQL scripts to test system

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

-- The unit tests for RHD DB.

--

-- First, load up the stk\_unit utility. This must be done as DB admin. E.g.:

--

-- sudo bash -c \

-- 'mysql -h localhost -D rhd < ../stk\_unit1.0-rc6/sql/stk\_unit.sql'

--

-- Then, read this file in as root.

-- had to run as system root => we won't be able to run this on the uni system

--

-- Run the test by issuing, as root:

--

-- CALL stk\_unit.tc('test\_rhd');

--

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

CONNECT mysql ;

DROP DATABASE IF EXISTS test\_rhd ;

CREATE DATABASE test\_rhd CHARACTER SET = utf8;

CONNECT test\_rhd;

DELIMITER $$

-- ensure that our populate scripts put at least some data in every table

CREATE PROCEDURE test\_populate\_contents()

BEGIN

DECLARE rowCount INT;

DECLARE tableName varchar(64) DEFAULT "";

DECLARE tablesDone INTEGER DEFAULT 0;

-- iterate over all the tables in RHD DB

DECLARE curs CURSOR FOR SELECT t.TABLE\_NAME from information\_schema.tables t

WHERE t.TABLE\_SCHEMA = 'rhd'

AND t.table\_type = 'BASE TABLE';

DECLARE CONTINUE HANDLER FOR NOT FOUND SET tablesDone = 1 ;

OPEN curs ;

test\_count: LOOP

FETCH curs INTO tableName ;

IF tablesDone = 1 THEN

LEAVE test\_count ;

END IF ;

SET @s = CONCAT('SELECT COUNT(\*) FROM rhd.`', tableName,

'` into @OUTVAR') ;

PREPARE rowCountStmt FROM @s ;

EXECUTE rowCountStmt;

SELECT @OUTVAR INTO rowCount;

CALL stk\_unit.assert\_false(rowCount = 0, CONCAT('Table has zero rows: ',

tableName)) ;

END LOOP test\_count ;

CLOSE curs ;

END $$

CREATE PROCEDURE test\_utrans\_a1()

BEGIN

-- run the query for this test and create a temporary table for results

CREATE TEMPORARY TABLE utrans\_a1\_actuals

SELECT ApplicantID

FROM rhd.Applicant

WHERE Applicant.FName = 'Shirin'

AND Applicant.LName = 'Ebadi' ;

-- create a temporary table to store the expected results

CREATE TEMPORARY TABLE utrans\_a1\_expecteds (

ApplicantID int(10) NOT NULL,

PRIMARY KEY (ApplicantID)

) ;

INSERT INTO utrans\_a1\_expecteds (ApplicantID)

VALUES (1);

-- create another temporary table that lists the differences between expected

-- results and actuals

CREATE TEMPORARY TABLE utrans\_a1\_problems

SELECT problems.ApplicantID

FROM

(

SELECT actuals.ApplicantID

FROM utrans\_a1\_actuals actuals

UNION ALL

SELECT expecteds.ApplicantID

FROM utrans\_a1\_expecteds expecteds

) problems

GROUP BY problems.ApplicantID

HAVING COUNT(\*) = 1

ORDER BY problems.ApplicantID ;

-- report an error if the 'problems' table isn't empty

CALL stk\_unit.assert\_table\_empty(

DATABASE(),

'utrans\_a1\_problems',

'Incorrect results') ;

DROP TABLE utrans\_a1\_expecteds ;

DROP TABLE utrans\_a1\_actuals ;

DROP TABLE utrans\_a1\_problems ;

END $$

CREATE PROCEDURE test\_utrans\_a2()

BEGIN

-- run the query for this test and create a temporary table for results

CREATE TEMPORARY TABLE utrans\_a2\_actuals

SELECT p.PubID FROM rhd.Publication p, rhd.Applicant a

WHERE p.ApplicantID = a.ApplicantID

AND a.FName = 'Abdul-Allah'

AND a.LName = 'Al-Sadhan' ;

-- create a temporary table to store the expected results

CREATE TEMPORARY TABLE utrans\_a2\_expecteds (

PubID int(10) NOT NULL,

PRIMARY KEY (PubID)

) ;

INSERT INTO utrans\_a2\_expecteds (PubID)

VALUES (2000);

-- create another temporary table that lists the differences between expected

-- results and actuals

CREATE TEMPORARY TABLE utrans\_a2\_problems

SELECT problems.PubID

FROM

(

SELECT actuals.PubID

FROM utrans\_a2\_actuals actuals

UNION ALL

SELECT expecteds.PubID

FROM utrans\_a2\_expecteds expecteds

) problems

GROUP BY problems.PubID

HAVING COUNT(\*) = 1

ORDER BY problems.PubID ;

-- report an error if the 'problems' table isn't empty

CALL stk\_unit.assert\_table\_empty(

DATABASE(),

'utrans\_a2\_problems',

'Incorrect results') ;

DROP TABLE utrans\_a2\_expecteds ;

DROP TABLE utrans\_a2\_actuals ;

DROP TABLE utrans\_a2\_problems ;

END $$

CREATE PROCEDURE test\_utrans\_a3()

BEGIN

-- run the query for this test and create a temporary table for results

CREATE TEMPORARY TABLE utrans\_a3\_actuals

SELECT d.DegID FROM rhd.Degree d, rhd.Applicant a

WHERE d.ApplicantID = a.ApplicantID

AND a.FName = 'Shirin'

AND a.LName = 'Ebadi' ;

-- create a temporary table to store the expected results

CREATE TEMPORARY TABLE utrans\_a3\_expecteds (

DegID int(10) NOT NULL,

PRIMARY KEY (DegID)

) ;

INSERT INTO utrans\_a3\_expecteds (DegID)

VALUES (121), (122);

-- create another temporary table that lists the differences between expected

-- results and actuals

CREATE TEMPORARY TABLE utrans\_a3\_problems

SELECT problems.DegID

FROM

(

SELECT actuals.DegID

FROM utrans\_a3\_actuals actuals

UNION ALL

SELECT expecteds.DegID

FROM utrans\_a3\_expecteds expecteds

) problems

GROUP BY problems.DegID

HAVING COUNT(\*) = 1

ORDER BY problems.DegID ;

-- report an error if the 'problems' table isn't empty

CALL stk\_unit.assert\_table\_empty(

DATABASE(),

'utrans\_a3\_problems',

'Incorrect results') ;

DROP TABLE utrans\_a3\_expecteds ;

DROP TABLE utrans\_a3\_actuals ;

DROP TABLE utrans\_a3\_problems ;

END $$

-- a4

CREATE PROCEDURE test\_utrans\_a4()

BEGIN

-- run the query for this test and create a temporary table for results

CREATE TEMPORARY TABLE utrans\_a4\_actuals

SELECT V.VisaID

FROM rhd.Visa V

INNER JOIN rhd.Applicant A ON V.ApplicantID = A.ApplicantID

WHERE A.FName = 'Shirin' AND A.LName = 'Ebadi' ;

-- create a temporary table to store the expected results

CREATE TEMPORARY TABLE utrans\_a4\_expecteds (

VisaID int(10) NOT NULL,

PRIMARY KEY (VisaID)

) ;

INSERT INTO utrans\_a4\_expecteds (VisaID)

VALUES (3001);

-- create another temporary table that lists the differences between expected

-- results and actuals

CREATE TEMPORARY TABLE utrans\_a4\_problems

SELECT problems.VisaID

FROM

(

SELECT actuals.VisaID

FROM utrans\_a4\_actuals actuals

UNION ALL

SELECT expecteds.VisaID

FROM utrans\_a4\_expecteds expecteds

) problems

GROUP BY problems.VisaID

HAVING COUNT(\*) = 1

ORDER BY problems.VisaID ;

-- report an error if the 'problems' table isn't empty

CALL stk\_unit.assert\_table\_empty(

DATABASE(),

'utrans\_a4\_problems',

'Incorrect results') ;

DROP TABLE utrans\_a4\_expecteds ;

DROP TABLE utrans\_a4\_actuals ;

DROP TABLE utrans\_a4\_problems ;

END $$

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

-- a5

CREATE PROCEDURE test\_utrans\_a5()

BEGIN

-- run the query for this test and create a temporary table for results

CREATE TEMPORARY TABLE utrans\_a5\_actuals

SELECT d.DocID, dt.Type AS DocType

FROM rhd.Applicant App

INNER JOIN rhd.Document d ON d.ApplicantID = App.ApplicantID

INNER JOIN rhd.`Document Type` dt ON d.DocTypeID = dt.DocTypeID

WHERE App.FName = 'Shirin'

AND App.LName = 'Ebadi' ;

-- create a temporary table to store the expected results

-- one row, id 131, cv

CREATE TEMPORARY TABLE utrans\_a5\_expecteds (

DocID int(10) NOT NULL,

DocType varchar(50)

) ;

INSERT INTO utrans\_a5\_expecteds (DocID, DocType)

VALUES (131, 'cv');

-- create another temporary table that lists the differences between expected

-- results and actuals

CREATE TEMPORARY TABLE utrans\_a5\_problems

SELECT problems.\*

FROM

(

SELECT actuals.\*

FROM utrans\_a5\_actuals actuals

UNION ALL

SELECT expecteds.\*

FROM utrans\_a5\_expecteds expecteds

) problems

GROUP BY problems.DocID

HAVING COUNT(\*) = 1

ORDER BY problems.DocID ;

-- report an error if the 'problems' table isn't empty

CALL stk\_unit.assert\_table\_empty(

DATABASE(),

'utrans\_a5\_problems',

'Incorrect results') ;

DROP TABLE utrans\_a5\_expecteds ;

DROP TABLE utrans\_a5\_actuals ;

DROP TABLE utrans\_a5\_problems ;

END $$

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

-- b) Look up applicant's applications by applicant name

CREATE PROCEDURE test\_utrans\_b()

BEGIN

-- run the query for this test and create a temporary table for results

CREATE TEMPORARY TABLE utrans\_b\_actuals

SELECT App.ApplicationID, AwType.Type AS awardType

FROM rhd.Applicant Appt

INNER JOIN rhd.Application App ON Appt.ApplicantID = App.ApplicantID

INNER JOIN rhd.`Award Type` AwType ON App.AwardID = AwType.AwardID

WHERE Appt.FName = 'Shirin'

AND Appt.LName = 'Ebadi' ;

-- create a temporary table to store the expected results

-- expect ApplicationID = 111, Type = 'PhD'

CREATE TEMPORARY TABLE utrans\_b\_expecteds (

ApplicationID int(10) NOT NULL,

awardType varchar(50)

) ;

INSERT INTO utrans\_b\_expecteds (ApplicationID, awardType)

VALUES (111, 'PhD');

-- create another temporary table that lists the differences between expected

-- results and actuals

CREATE TEMPORARY TABLE utrans\_b\_problems

SELECT problems.\*

FROM

(

SELECT actuals.\*

FROM utrans\_b\_actuals actuals

UNION ALL

SELECT expecteds.\*

FROM utrans\_b\_expecteds expecteds

) problems

GROUP BY problems.ApplicationID

HAVING COUNT(\*) = 1

ORDER BY problems.ApplicationID ;

-- report an error if the 'problems' table isn't empty

CALL stk\_unit.assert\_table\_empty(

DATABASE(),

'utrans\_b\_problems',

'Incorrect results') ;

DROP TABLE utrans\_b\_expecteds ;

DROP TABLE utrans\_b\_actuals ;

DROP TABLE utrans\_b\_problems ;

END $$

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

-- c) Look up applicant's applications by applicant email

CREATE PROCEDURE test\_utrans\_c()

BEGIN

-- run the query for this test and create a temporary table for results

CREATE TEMPORARY TABLE utrans\_c\_actuals

SELECT App.ApplicationID, AwType.Type AS awardType

FROM rhd.Applicant Appt

INNER JOIN rhd.Application App ON Appt.ApplicantID = App.ApplicantID

INNER JOIN rhd.`Award Type` AwType ON App.AwardID = AwType.AwardID

WHERE Appt.Email = 'don.memo@hotmail.com' ;

-- create a temporary table to store the expected results

-- expect ApplicationID = 211, Type = 'PhD'

CREATE TEMPORARY TABLE utrans\_c\_expecteds (

ApplicationID int(10) NOT NULL,

awardType varchar(50)

) ;

INSERT INTO utrans\_c\_expecteds (ApplicationID, awardType)

VALUES (211, 'PhD');

-- create another temporary table that lists the differences between expected

-- results and actuals

CREATE TEMPORARY TABLE utrans\_c\_problems

SELECT problems.\*

FROM

(

SELECT actuals.\*

FROM utrans\_c\_actuals actuals

UNION ALL

SELECT expecteds.\*

FROM utrans\_c\_expecteds expecteds

) problems

GROUP BY problems.ApplicationID

HAVING COUNT(\*) = 1

ORDER BY problems.ApplicationID ;

-- report an error if the 'problems' table isn't empty

CALL stk\_unit.assert\_table\_empty(

DATABASE(),

'utrans\_c\_problems',

'Incorrect results') ;

DROP TABLE utrans\_c\_expecteds ;

DROP TABLE utrans\_c\_actuals ;

DROP TABLE utrans\_c\_problems ;

END $$

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

-- d) Look up incomplete applications

CREATE PROCEDURE test\_utrans\_d()

BEGIN

-- run the query for this test and create a temporary table for results

CREATE TEMPORARY TABLE utrans\_d\_actuals

SELECT App.ApplicationID

FROM rhd.Application App

WHERE App.applicationStatusID < 10500 ;

-- create a temporary table to store the expected results

-- expect all 11 applications

CREATE TEMPORARY TABLE utrans\_d\_expecteds (

ApplicationID int(10) NOT NULL

) ;

INSERT INTO utrans\_d\_expecteds (ApplicationID)

VALUES (111), (211), (311), (411), (511), (611), (711), (811), (911), (1011),

(1111) ;

-- create another temporary table that lists the differences between expected

-- results and actuals

CREATE TEMPORARY TABLE utrans\_d\_problems

SELECT problems.\*

FROM

(

SELECT actuals.\*

FROM utrans\_d\_actuals actuals

UNION ALL

SELECT expecteds.\*

FROM utrans\_d\_expecteds expecteds

) problems

GROUP BY problems.ApplicationID

HAVING COUNT(\*) = 1

ORDER BY problems.ApplicationID ;

-- report an error if the 'problems' table isn't empty

CALL stk\_unit.assert\_table\_empty(

DATABASE(),

'utrans\_d\_problems',

'Incorrect results') ;

DROP TABLE utrans\_d\_expecteds ;

DROP TABLE utrans\_d\_actuals ;

DROP TABLE utrans\_d\_problems ;

END $$

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

-- e) Look up all correspondences relevant to an application

CREATE PROCEDURE test\_utrans\_e()

BEGIN

-- run the query for this test and create a temporary table for results

CREATE TEMPORARY TABLE utrans\_e\_actuals

SELECT Corr.CorrID

FROM rhd.Application App

INNER JOIN rhd.Correspondence Corr ON Corr.ApplicationID = App.ApplicationID

WHERE App.ApplicationID = 411 ;

-- create a temporary table to store the expected results

-- two rows, id 441 and 442,

CREATE TEMPORARY TABLE utrans\_e\_expecteds (

CorrID int(10) NOT NULL

) ;

INSERT INTO utrans\_e\_expecteds (CorrID)

VALUES (441), (442) ;

-- create another temporary table that lists the differences between expected

-- results and actuals

CREATE TEMPORARY TABLE utrans\_e\_problems

SELECT problems.\*

FROM

(

SELECT actuals.\*

FROM utrans\_e\_actuals actuals

UNION ALL

SELECT expecteds.\*

FROM utrans\_e\_expecteds expecteds

) problems

GROUP BY problems.CorrID

HAVING COUNT(\*) = 1

ORDER BY problems.CorrID ;

-- report an error if the 'problems' table isn't empty

CALL stk\_unit.assert\_table\_empty(

DATABASE(),

'utrans\_e\_problems',

'Incorrect results') ;

DROP TABLE utrans\_e\_expecteds ;

DROP TABLE utrans\_e\_actuals ;

DROP TABLE utrans\_e\_problems ;

END $$

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

-- g) Look up which staff member updated an Application most recently

CREATE PROCEDURE test\_utrans\_g()

BEGIN

-- run the query for this test and create a temporary table for results

CREATE TEMPORARY TABLE utrans\_g\_actuals

SELECT USM.StaffID, USM.FName, USM.LName

FROM rhd.Application App

INNER JOIN rhd.`University Staff Member` USM

ON USM.StaffID = App.LastModifiedByStaffID

WHERE App.ApplicationID = 311 ;

-- create a temporary table to store the expected results

-- expect 'Paul' 'Calder'

CREATE TEMPORARY TABLE utrans\_g\_expecteds (

StaffID int(10),

FName varchar(50),

LName varchar(50)

) ;

INSERT INTO utrans\_g\_expecteds (StaffID, FName, LName)

VALUES (1001, 'Paul', 'Calder') ;

-- create another temporary table that lists the differences between expected

-- results and actuals

CREATE TEMPORARY TABLE utrans\_g\_problems

SELECT problems.\*

FROM

(

SELECT actuals.\*

FROM utrans\_g\_actuals actuals

UNION ALL

SELECT expecteds.\*

FROM utrans\_g\_expecteds expecteds

) problems

GROUP BY problems.StaffID

HAVING COUNT(\*) = 1

ORDER BY problems.StaffID ;

-- report an error if the 'problems' table isn't empty

CALL stk\_unit.assert\_table\_empty(

DATABASE(),

'utrans\_g\_problems',

'Incorrect results') ;

DROP TABLE utrans\_g\_expecteds ;

DROP TABLE utrans\_g\_actuals ;

DROP TABLE utrans\_g\_problems ;

END $$

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

-- h) Check for any decision recorded about an application

CREATE PROCEDURE test\_utrans\_h()

BEGIN

-- run the query for this test and create a temporary table for results

CREATE TEMPORARY TABLE utrans\_h\_actuals

SELECT Decision.StaffID, DT.type

FROM rhd.Application App

INNER JOIN rhd.Decision ON Decision.ApplicationID = App.ApplicationID

INNER JOIN rhd.`Decision Type` DT

ON DT.DecisionTypeID = Decision.DecisionTypeID

WHERE App.ApplicationID = 411 ;

-- create a temporary table to store the expected results

-- expect COUNT = 1

-- expect StaffID = 1001

-- expect dectype = 'RFI'

CREATE TEMPORARY TABLE utrans\_h\_expecteds (

StaffID int(10),

type varchar(50)

) ;

INSERT INTO utrans\_h\_expecteds (StaffID, type)

VALUES (1001, 'RFI' ) ;

-- create another temporary table that lists the differences between expected

-- results and actuals

CREATE TEMPORARY TABLE utrans\_h\_problems

SELECT problems.\*

FROM

(

SELECT actuals.\*

FROM utrans\_h\_actuals actuals

UNION ALL

SELECT expecteds.\*

FROM utrans\_h\_expecteds expecteds

) problems

GROUP BY problems.StaffID

HAVING COUNT(\*) = 1

ORDER BY problems.StaffID ;

-- report an error if the 'problems' table isn't empty

CALL stk\_unit.assert\_table\_empty(

DATABASE(),

'utrans\_h\_problems',

'Incorrect results') ;

DROP TABLE utrans\_h\_expecteds ;

DROP TABLE utrans\_h\_actuals ;

DROP TABLE utrans\_h\_problems ;

END $$

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

-- k) Look up an existing application and list outstanding information

-- (checklist).

CREATE PROCEDURE test\_utrans\_k()

BEGIN

-- run the query for this test and create a temporary table for results

CREATE TEMPORARY TABLE utrans\_k\_actuals

SELECT App.ApplicationID, AppStat.Status,

App.AddressConfirmed, App.DegreeConfirmed,

App.VisaStatusConfirmed, App.ProposalConfirmed, App.HasResearchAreas,

App.HasPrimarySuper, App.PayMethConfirmed, App.EngProfConfirmed

FROM rhd.Application App

INNER JOIN rhd.`Application Status` AppStat

ON AppStat.ApplicationStatusID = App.ApplicationStatusID

WHERE App.ApplicationID = 611 ;

-- create a temporary table to store the expected results

-- expect count == 1; status=ongoing, others 0/false

CREATE TEMPORARY TABLE utrans\_k\_expecteds (

ApplicationID int(10),

Status varchar(50),

AddressConfirmed int(1),

DegreeConfirmed int(1),

VisaStatusConfirmed int(1),

ProposalConfirmed int(1),

HasResearchAreas int(1),

HasPrimarySuper int(1),

PayMethConfirmed int(1),

EngProfConfirmed int(1)

) ;

INSERT INTO utrans\_k\_expecteds

VALUES (611, 'ongoing', 0, 0, 0, 0, 0, 0, 0, 0 ) ;

-- create another temporary table that lists the differences between expected

-- results and actuals

CREATE TEMPORARY TABLE utrans\_k\_problems

SELECT problems.\*

FROM

(

SELECT actuals.\*

FROM utrans\_k\_actuals actuals

UNION ALL

SELECT expecteds.\*

FROM utrans\_k\_expecteds expecteds

) problems

GROUP BY problems.ApplicationID

HAVING COUNT(\*) = 1

ORDER BY problems.ApplicationID ;

-- report an error if the 'problems' table isn't empty

CALL stk\_unit.assert\_table\_empty(

DATABASE(),

'utrans\_k\_problems',

'Incorrect results') ;

DROP TABLE utrans\_k\_expecteds ;

DROP TABLE utrans\_k\_actuals ;

DROP TABLE utrans\_k\_problems ;

END $$

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

-- l) Update the checklist to confirm that a mandatory information requirement

-- has been met

CREATE PROCEDURE test\_utrans\_l()

BEGIN

UPDATE rhd.Application

SET AddressConfirmed = 1

WHERE Application.ApplicationID = 611 ;

-- expect success

-- rerun k), expect AddressConfirmed == 1/true

-- run the query for this test and create a temporary table for results

CREATE TEMPORARY TABLE utrans\_l\_actuals

SELECT App.ApplicationID, AppStat.Status,

App.AddressConfirmed, App.DegreeConfirmed,

App.VisaStatusConfirmed, App.ProposalConfirmed, App.HasResearchAreas,

App.HasPrimarySuper, App.PayMethConfirmed, App.EngProfConfirmed

FROM rhd.Application App

INNER JOIN rhd.`Application Status` AppStat

ON AppStat.ApplicationStatusID = App.ApplicationStatusID

WHERE App.ApplicationID = 611 ;

-- create a temporary table to store the expected results

-- expect count == 1; status=ongoing, others 0/false

CREATE TEMPORARY TABLE utrans\_l\_expecteds (

ApplicationID int(10),

Status varchar(50),

AddressConfirmed int(1),

DegreeConfirmed int(1),

VisaStatusConfirmed int(1),

ProposalConfirmed int(1),

HasResearchAreas int(1),

HasPrimarySuper int(1),

PayMethConfirmed int(1),

EngProfConfirmed int(1)

) ;

INSERT INTO utrans\_l\_expecteds

VALUES (611, 'ongoing', 1, 0, 0, 0, 0, 0, 0, 0 ) ;

-- create another temporary table that lists the differences between expected

-- results and actuals

CREATE TEMPORARY TABLE utrans\_l\_problems

SELECT problems.\*

FROM

(

SELECT actuals.\*

FROM utrans\_l\_actuals actuals

UNION ALL

SELECT expecteds.\*

FROM utrans\_l\_expecteds expecteds

) problems

GROUP BY problems.ApplicationID

HAVING COUNT(\*) = 1

ORDER BY problems.ApplicationID ;

-- report an error if the 'problems' table isn't empty

CALL stk\_unit.assert\_table\_empty(

DATABASE(),

'utrans\_l\_problems',

'Incorrect results') ;

DROP TABLE utrans\_l\_expecteds ;

DROP TABLE utrans\_l\_actuals ;

DROP TABLE utrans\_l\_problems ;

-- reset to initial state

UPDATE rhd.Application

SET AddressConfirmed = 1

WHERE Application.ApplicationID = 611 ;

END $$

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

-- m) Retrieve all on-going applications for which the user has made the most

-- recent correspondence

CREATE PROCEDURE test\_utrans\_m()

BEGIN

-- run the query for this test and create a temporary table for results

CREATE TEMPORARY TABLE utrans\_m\_actuals

SELECT App.ApplicationID

FROM rhd.Application App

INNER JOIN rhd.`Application Status` AppStat

ON AppStat.ApplicationStatusID = App.ApplicationStatusID

WHERE AppStat.Status LIKE 'ongoing%'

AND App.LastModifiedByStaffID = 1002 ;

-- create a temporary table to store the expected results

-- three rows, ids 111, 611 and 1011

CREATE TEMPORARY TABLE utrans\_m\_expecteds (

ApplicationID int(10)

) ;

INSERT INTO utrans\_m\_expecteds

VALUES (111), (611), (1011) ;

-- create another temporary table that lists the differences between expected

-- results and actuals

CREATE TEMPORARY TABLE utrans\_m\_problems

SELECT problems.\*

FROM

(

SELECT actuals.\*

FROM utrans\_m\_actuals actuals

UNION ALL

SELECT expecteds.\*

FROM utrans\_m\_expecteds expecteds

) problems

GROUP BY problems.ApplicationID

HAVING COUNT(\*) = 1

ORDER BY problems.ApplicationID ;

-- report an error if the 'problems' table isn't empty

CALL stk\_unit.assert\_table\_empty(

DATABASE(),

'utrans\_m\_problems',

'Incorrect results') ;

DROP TABLE utrans\_m\_expecteds ;

DROP TABLE utrans\_m\_actuals ;

DROP TABLE utrans\_m\_problems ;

END $$

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

-- o) Update the status of an application

CREATE PROCEDURE test\_utrans\_o()

BEGIN

-- change the status of an application

UPDATE rhd.Application App

JOIN

(SELECT \* FROM rhd.`Application Status` AppStat

WHERE AppStat.Status = 'complete.declined') AS Lookup

SET App.applicationStatusID = Lookup.ApplicationStatusID

WHERE App.ApplicationID = 611 ;

-- run the query for this test and create a temporary table for results

CREATE TEMPORARY TABLE utrans\_o\_actuals

SELECT AppStat.Status

FROM rhd.`Application Status` AppStat

INNER JOIN rhd.Application App

ON AppStat.ApplicationStatusID = App.ApplicationStatusID

WHERE App.ApplicationID = 611;

-- create a temporary table to store the expected results

-- one row, 'complete.declined'

CREATE TEMPORARY TABLE utrans\_o\_expecteds (

Status varchar(50)

) ;

INSERT INTO utrans\_o\_expecteds

VALUES ('complete.declined');

-- create another temporary table that lists the differences between expected

-- results and actuals

CREATE TEMPORARY TABLE utrans\_o\_problems

SELECT problems.\*

FROM

(

SELECT actuals.\*

FROM utrans\_o\_actuals actuals

UNION ALL

SELECT expecteds.\*

FROM utrans\_o\_expecteds expecteds

) problems

GROUP BY problems.Status

HAVING COUNT(\*) = 1

ORDER BY problems.Status ;

-- report an error if the 'problems' table isn't empty

CALL stk\_unit.assert\_table\_empty(

DATABASE(),

'utrans\_o\_problems',

'Incorrect results') ;

DROP TABLE utrans\_o\_expecteds ;

DROP TABLE utrans\_o\_actuals ;

DROP TABLE utrans\_o\_problems ;

-- reset the status of an application

UPDATE rhd.Application App

JOIN

(SELECT \* FROM rhd.`Application Status` AppStat

WHERE AppStat.Status = 'ongoing') AS Lookup

SET App.applicationStatusID = Lookup.ApplicationStatusID

WHERE App.ApplicationID = 611 ;

END $$

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

-- p) Look up, add to, and delete from own current research areas

-- p1) look up current research areas

CREATE PROCEDURE test\_utrans\_p1()

BEGIN

-- run the query for this test and create a temporary table for results

CREATE TEMPORARY TABLE utrans\_p1\_actuals

SELECT USM.FORCode

FROM rhd.`University Staff Member\_Research Area` USM

WHERE USM.StaffID = 1000 ;

-- create a temporary table to store the expected results

-- expect count == 4, 80399, 80499, 80699, 89999

CREATE TEMPORARY TABLE utrans\_p1\_expecteds (

FORCode int(10)

) ;

INSERT INTO utrans\_p1\_expecteds

VALUES (80399), (80499), (80699), (89999) ;

-- create another temporary table that lists the differences between expected

-- results and actuals

CREATE TEMPORARY TABLE utrans\_p1\_problems

SELECT problems.\*

FROM

(

SELECT actuals.\*

FROM utrans\_p1\_actuals actuals

UNION ALL

SELECT expecteds.\*

FROM utrans\_p1\_expecteds expecteds

) problems

GROUP BY problems.FORCode

HAVING COUNT(\*) = 1

ORDER BY problems.FORCode ;

-- report an error if the 'problems' table isn't empty

CALL stk\_unit.assert\_table\_empty(

DATABASE(),

'utrans\_p1\_problems',

'Incorrect results') ;

DROP TABLE utrans\_p1\_expecteds ;

DROP TABLE utrans\_p1\_actuals ;

DROP TABLE utrans\_p1\_problems ;

END $$

-- p2) Insert new research areas tested in populate script

-- p3) Delete research areas

CREATE PROCEDURE test\_utrans\_p3()

BEGIN

-- remove a research area

DELETE rhd.`University Staff Member\_Research Area` USMRA

FROM rhd.`University Staff Member\_Research Area` USMRA

JOIN

(SELECT RA.FORCode FROM rhd.`Research Area` RA

WHERE RA.Description LIKE 'information systems not elsewhere classified')

AS Lookup

WHERE StaffID = 1000

AND USMRA.FORCode = Lookup.FORCode;

-- run the query for this test and create a temporary table for results

CREATE TEMPORARY TABLE utrans\_p3\_actuals

SELECT USM.FORCode

FROM rhd.`University Staff Member\_Research Area` USM

WHERE USM.StaffID = 1000 ;

-- create a temporary table to store the expected results

-- expect count == 4, 80399, 80499, 89999

CREATE TEMPORARY TABLE utrans\_p3\_expecteds (

FORCode int(10)

) ;

INSERT INTO utrans\_p3\_expecteds

VALUES (80399), (80499), (89999) ;

-- create another temporary table that lists the differences between expected

-- results and actuals

CREATE TEMPORARY TABLE utrans\_p3\_problems

SELECT problems.\*

FROM

(

SELECT actuals.\*

FROM utrans\_p3\_actuals actuals

UNION ALL

SELECT expecteds.\*

FROM utrans\_p3\_expecteds expecteds

) problems

GROUP BY problems.FORCode

HAVING COUNT(\*) = 1

ORDER BY problems.FORCode ;

-- report an error if the 'problems' table isn't empty

CALL stk\_unit.assert\_table\_empty(

DATABASE(),

'utrans\_p3\_problems',

'Incorrect results') ;

DROP TABLE utrans\_p3\_expecteds ;

DROP TABLE utrans\_p3\_actuals ;

DROP TABLE utrans\_p3\_problems ;

-- Re-instate the temporarily removed research area

INSERT INTO rhd.`University Staff Member\_Research Area`

(StaffID, FORCode)

(SELECT USM.StaffID, Lookup.FORCode FROM rhd.`University Staff Member` USM

JOIN

(SELECT RA.FORCode FROM rhd.`Research Area` RA

WHERE RA.Description LIKE 'information systems not elsewhere classified')

AS Lookup

WHERE USM.StaffID = 1000 );

END $$

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

-- q) Search for all applications in certain research areas that have been added

-- since a certain time

CREATE PROCEDURE test\_utrans\_q()

BEGIN

-- run the query for this test and create a temporary table for results

CREATE TEMPORARY TABLE utrans\_q\_actuals

SELECT App.ApplicationID

FROM rhd.Application App

INNER JOIN rhd.`Application\_Research Area` ARA

ON App.ApplicationID = ARA.ApplicationID

WHERE App.DateAdded >= '2014-05-01'

AND ARA.FORCode = 100503 ;

-- create a temporary table to store the expected results

-- expect 1 row, id 611

CREATE TEMPORARY TABLE utrans\_q\_expecteds (

ApplicationID int(10)

) ;

INSERT INTO utrans\_q\_expecteds

VALUES (611), (811) ;

-- create another temporary table that lists the differences between expected

-- results and actuals

CREATE TEMPORARY TABLE utrans\_q\_problems

SELECT problems.\*

FROM

(

SELECT actuals.\*

FROM utrans\_q\_actuals actuals

UNION ALL

SELECT expecteds.\*

FROM utrans\_q\_expecteds expecteds

) problems

GROUP BY problems.ApplicationID

HAVING COUNT(\*) = 1

ORDER BY problems.ApplicationID ;

-- report an error if the 'problems' table isn't empty

CALL stk\_unit.assert\_table\_empty(

DATABASE(),

'utrans\_q\_problems',

'Incorrect results') ;

DROP TABLE utrans\_q\_expecteds ;

DROP TABLE utrans\_q\_actuals ;

DROP TABLE utrans\_q\_problems ;

END $$

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

-- s) Retrieve all staff who have flagged an application, or have edited an

-- application or applicant record most recently

--

-- s1) Retrieve the flagged applications

CREATE PROCEDURE test\_utrans\_s1()

BEGIN

-- run the query for this test and create a temporary table for results

CREATE TEMPORARY TABLE utrans\_s1\_actuals

SELECT USM.StaffID, USM.FName, USM.LName

FROM rhd.`University Staff Member` USM

INNER JOIN rhd.`University Staff Member\_Application` Flags

ON Flags.StaffID = USM.StaffID

WHERE Flags.ApplicationID = 711 ;

-- create a temporary table to store the expected results

-- expect Denise de Vries and John Roddick

CREATE TEMPORARY TABLE utrans\_s1\_expecteds (

StaffID int(10),

FName varchar(50),

LName varchar(50)

) ;

INSERT INTO utrans\_s1\_expecteds

VALUES (1002, 'John', 'Roddick'), (1000, 'Denise', 'de Vries') ;

-- create another temporary table that lists the differences between expected

-- results and actuals

CREATE TEMPORARY TABLE utrans\_s1\_problems

SELECT problems.\*

FROM

(

SELECT actuals.\*

FROM utrans\_s1\_actuals actuals

UNION ALL

SELECT expecteds.\*

FROM utrans\_s1\_expecteds expecteds

) problems

GROUP BY problems.StaffID

HAVING COUNT(\*) = 1

ORDER BY problems.StaffID ;

-- report an error if the 'problems' table isn't empty

CALL stk\_unit.assert\_table\_empty(

DATABASE(),

'utrans\_s1\_problems',

'Incorrect results') ;

DROP TABLE utrans\_s1\_expecteds ;

DROP TABLE utrans\_s1\_actuals ;

DROP TABLE utrans\_s1\_problems ;

END $$

-- s2) Last staff member to modify an application

CREATE PROCEDURE test\_utrans\_s2()

BEGIN

-- run the query for this test and create a temporary table for results

CREATE TEMPORARY TABLE utrans\_s2\_actuals

SELECT USM.StaffID, USM.FName, USM.LName

FROM rhd.`University Staff Member` USM

INNER JOIN rhd.Application App

ON App.LastModifiedByStaffID = USM.StaffID

WHERE App.ApplicationID = 711 ;

-- create a temporary table to store the expected results

-- expect Denise de Vries and John Roddick

CREATE TEMPORARY TABLE utrans\_s2\_expecteds (

StaffID int(10),

FName varchar(50),

LName varchar(50)

) ;

INSERT INTO utrans\_s2\_expecteds

VALUES (1000, 'Denise', 'de Vries') ;

-- create another temporary table that lists the differences between expected

-- results and actuals

CREATE TEMPORARY TABLE utrans\_s2\_problems

SELECT problems.\*

FROM

(

SELECT actuals.\*

FROM utrans\_s2\_actuals actuals

UNION ALL

SELECT expecteds.\*

FROM utrans\_s2\_expecteds expecteds

) problems

GROUP BY problems.StaffID

HAVING COUNT(\*) = 1

ORDER BY problems.StaffID ;

-- report an error if the 'problems' table isn't empty

CALL stk\_unit.assert\_table\_empty(

DATABASE(),

'utrans\_s2\_problems',

'Incorrect results') ;

DROP TABLE utrans\_s2\_expecteds ;

DROP TABLE utrans\_s2\_actuals ;

DROP TABLE utrans\_s2\_problems ;

END $$

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

-- t) Retrieve all ongoing applications

CREATE PROCEDURE test\_utrans\_t()

BEGIN

-- run the query for this test and create a temporary table for results

CREATE TEMPORARY TABLE utrans\_t\_actuals

SELECT App.ApplicationID

FROM rhd.Application App

INNER JOIN rhd.`Application Status` AppStat

ON App.ApplicationStatusID = AppStat.ApplicationStatusID

WHERE AppStat.Status LIKE 'ongoing%' ;

-- create a temporary table to store the expected results

-- expect all applications at this stage

CREATE TEMPORARY TABLE utrans\_t\_expecteds (

ApplicationID int(10)

) ;

INSERT INTO utrans\_t\_expecteds

VALUES (111), (211), (311), (411), (511), (611), (711), (811), (911), (1011),

(1111) ;

-- create another temporary table that lists the differences between expected

-- results and actuals

CREATE TEMPORARY TABLE utrans\_t\_problems

SELECT problems.\*

FROM

(

SELECT actuals.\*

FROM utrans\_t\_actuals actuals

UNION ALL

SELECT expecteds.\*

FROM utrans\_t\_expecteds expecteds

) problems

GROUP BY problems.ApplicationID

HAVING COUNT(\*) = 1

ORDER BY problems.ApplicationID ;

-- report an error if the 'problems' table isn't empty

CALL stk\_unit.assert\_table\_empty(

DATABASE(),

'utrans\_t\_problems',

'Incorrect results') ;

DROP TABLE utrans\_t\_expecteds ;

DROP TABLE utrans\_t\_actuals ;

DROP TABLE utrans\_t\_problems ;

END $$

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

-- u) List all applications waiting for supervisor agreement

CREATE PROCEDURE test\_utrans\_u()

BEGIN

-- run the query for this test and create a temporary table for results

CREATE TEMPORARY TABLE utrans\_u\_actuals

SELECT Agg.ApplicationID FROM

(SELECT

App.ApplicationID,

Super.StaffID,

Super.PrimarySupervisor,

SUM(Super.PrimarySupervisor) AS PriSuperSum

FROM rhd.Application App

LEFT OUTER JOIN rhd.`Supervise as` Super

ON Super.ApplicationID = App.ApplicationID

GROUP BY App.ApplicationID)

AS Agg

WHERE Agg.PriSuperSum <> 1

OR Agg.PriSuperSum IS NULL ;

-- create a temporary table to store the expected results

-- expect all applications except 111 and 311

CREATE TEMPORARY TABLE utrans\_u\_expecteds (

ApplicationID int(10)

) ;

INSERT INTO utrans\_u\_expecteds

VALUES (211), (411), (511), (611), (711), (811), (911), (1011), (1111) ;

-- create another temporary table that lists the differences between expected

-- results and actuals

CREATE TEMPORARY TABLE utrans\_u\_problems

SELECT problems.\*

FROM

(

SELECT actuals.\*

FROM utrans\_u\_actuals actuals

UNION ALL

SELECT expecteds.\*

FROM utrans\_u\_expecteds expecteds

) problems

GROUP BY problems.ApplicationID

HAVING COUNT(\*) = 1

ORDER BY problems.ApplicationID ;

-- report an error if the 'problems' table isn't empty

CALL stk\_unit.assert\_table\_empty(

DATABASE(),

'utrans\_u\_problems',

'Incorrect results') ;

DROP TABLE utrans\_u\_expecteds ;

DROP TABLE utrans\_u\_actuals ;

DROP TABLE utrans\_u\_problems ;

END $$

DELIMITER ;

# 

We ran the test code listed above in section 11 using the STK ([Picchiarelli and Razzoli 2013](#_ENREF_2" \o "Picchiarelli, 2013 #6)). Each of these tests defines the query to run and the expected results. It then automatically compares the actual results with the expected results and logs any differences. Below is a report of running these tests. Note that due to the restrictions of the university system, we ran these tests on other MySQL servers with which we had more control.

+------------------------------------------------------------------------------+

| report |

+------------------------------------------------------------------------------+

|

Test Case: test\_rhd

Id: 25

Completed: YES

47 passes, 0 fails, 0 exceptions

|

+------------------------------------------------------------------------------+

# References

Oracle Inc. (2014). "MySQL 5.1 Reference Manual - Chapter 14 Storage Engines." from <http://dev.mysql.com/doc/refman/5.1/en/storage-engines.html>.

Picchiarelli, G. and F. Razzoli (2013). "STK/Unit." Retrieved 2014.04.09, from <http://stk.wikidot.com/stk-unit>.

1. <http://dev.mysql.com/doc/refman/5.7/en/storage-engines.html> [↑](#footnote-ref-1)
2. <http://dev.mysql.com/doc/refman/5.7/en/innodb-index-types.html> [↑](#footnote-ref-2)