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 2](#_Toc388077020)

[1.1. Select target DBMS 2](#_Toc388077021)

[1.2. Design base relations 2](#_Toc388077022)

[1.3. Design representation of derived data 2](#_Toc388077023)

[1.4. Design general constraints 2](#_Toc388077024)

[2. Design file organisations and indices 2](#_Toc388077025)

[2.1. Analyse transactions 2](#_Toc388077026)

[2.2. Select file organisations 2](#_Toc388077027)

[2.3. Select indices 2](#_Toc388077028)

[2.4. Estimate disk space requirements 2](#_Toc388077029)

[3. Design user views 2](#_Toc388077030)

[4. Design security mechanisms 2](#_Toc388077031)

[5. Introduce controlled redundancy if necessary 2](#_Toc388077032)

[6. Create SQL scripts for data definition 2](#_Toc388077033)

[7. Create SQL scripts to populate all tables with data 2](#_Toc388077034)

[8. Create SQL scripts for required queries 2](#_Toc388077035)

[9. Monitor and tune the operational system 2](#_Toc388077036)

[10. Update test plan 2](#_Toc388077037)

[11. Create SQL scripts to test system 2](#_Toc388077038)

[12. Test operational system 2](#_Toc388077039)

# Questions

# 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 NOT DONE

The main transactions of the database, those that have a high impact, run frequently and or are critical to creating and updating RHD application and applicant details have been analysed. The transactions as described by the transaction pathways section of the logical documentation, section??? have been used to produce map the transactional pathways to the relations.

1. map all transaction paths to relations;

logical 3.0 plus more

1. determine which relations are most frequently accessed by transactions;

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.

Note that it is assumed that the staff ID will be cached by the application interface reducing impact on looking themselves up in the staff table every session.

1. Analyse the data usage of selected transactions that involve these relations.

## Select file organisations DONE

The file organisations are grouped by storage engines in MySQL[[1]](#footnote-1). The default InnoDB storage engine provides the required functionality for all relations, the other storage engines are 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 SQL scripts to populate all tables with data

# Create SQL scripts for required queries

# Monitor and tune the operational system

Possible performance enhancements

# Update test plan

# Create SQL scripts to test system

# Test operational system

The following list is no longer needed

* Publication
* Document
* Degree
* VISA
* Correspondence
* Application
* Referee
* Decision
* Research Area
* University staff

**Pair Relations**

* Application Research Area
* Supervise as
* University\_Staff\_Member\_Research\_Area
* University\_Staff\_Member\_Research\_Area2
* University\_Staff\_Member\_Applicaiton

**Lookup Relations**

* Document Type
* Document Status
* Visa Status
* Country
* Correspondence Method
* Payment Method
* Application Status
* Award Type
* Decision

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)