Constructing SQL Queries for AtoM

An overview of AtoM's data model and start constructing simple queries for reporting and data cleanup, among other uses, using MySQL Workbench.

Outline

Utilities to ease working with MySQL

Data model overview and resources

Explore the schema using SQL queries

Practical examples

MySQL Client Utilities

MySQL command line interface

- Req command line access

MySQL Workbench

- Runs locally
- Network connection to db

PHPMyAdmin

- Web delivered
- Requires installation on server

Sequel Pro (for macOS)

Today I'll be using **MySQL Workbench**

- Windows, macOS (OS X), Linux clients
- Don't really want to install anything directly on Vagrant box as I purge it frequently

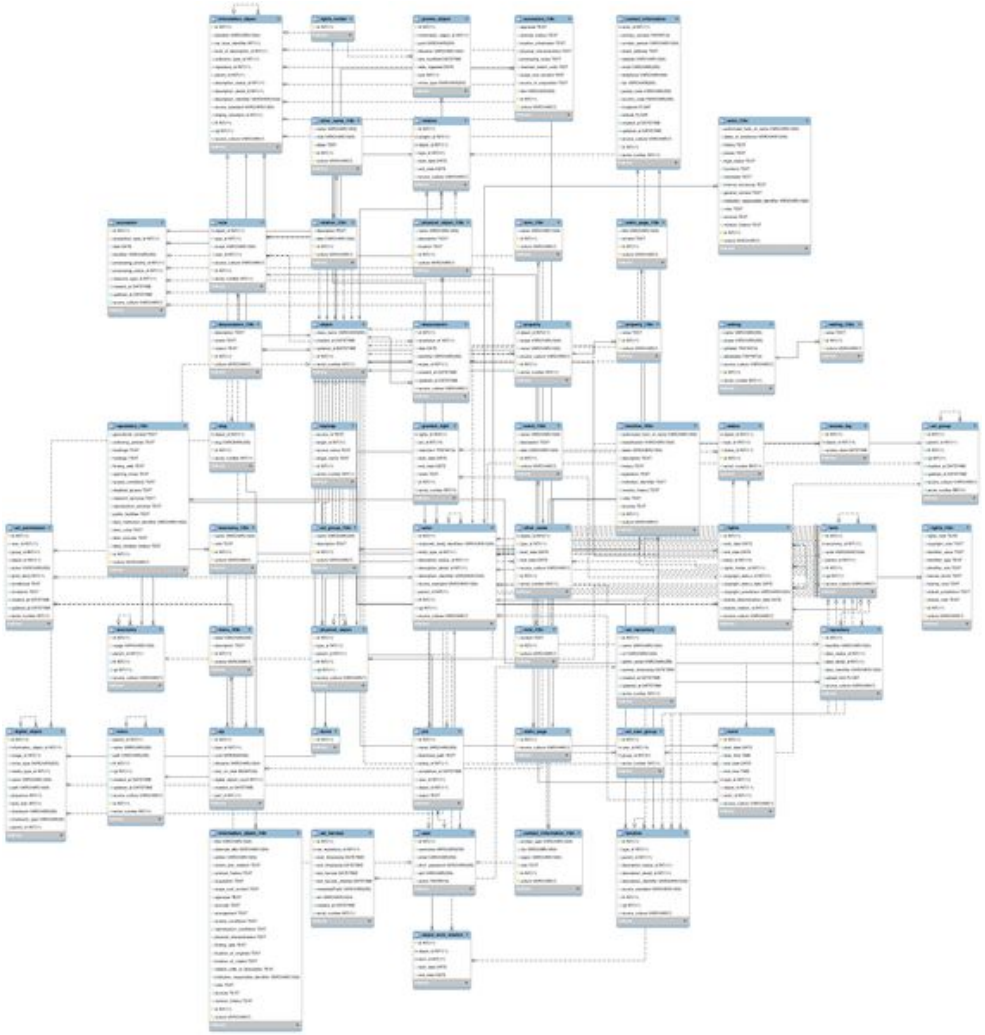
MySQL Workbench

Installing for use with AtoM Vagrant box

- Download from:
  - https://www.mysql.com/products/workbench/
  - ...and run the installer
- Grant access to a user to connect to mysql from host machine
  - `mysql -u root -h localhost -p`
  - `GRANT ALL ON *.* to root@'10.10.10.1' IDENTIFIED BY 'root';`
  - `FLUSH PRIVILEGES;`
- Launch MySQL Workbench
  - Connect to `10.10.10.10`
  - User: `root`
  - Pw: `root`

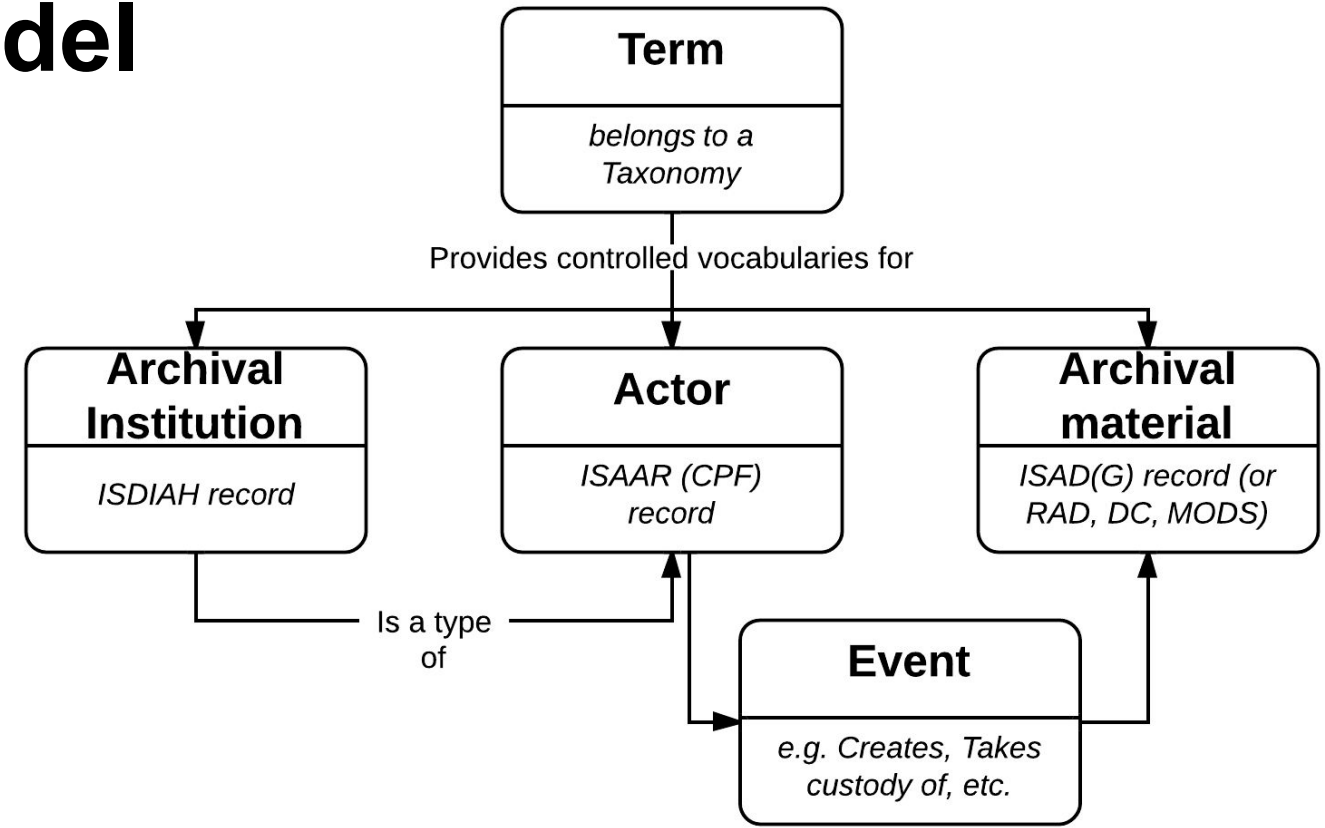
AtoM's ERD

We are going to focus on a few specific tables



https://wiki.accesstomemory.org/Development/ERDs

Entity Model



https://www.accesstomemory.org/en/docs/2.3/user-manual/overview/entity-types/

Examine an Archival Description

Tables:

- `slug`
- `information_object`

Let's look at an Archival description in AtoM:

- Fred Wah Fonds
- Examine the URL:
- `http://10.10.10.10/fred-wah-fonds`
- Slug is "fred-wah-fonds"

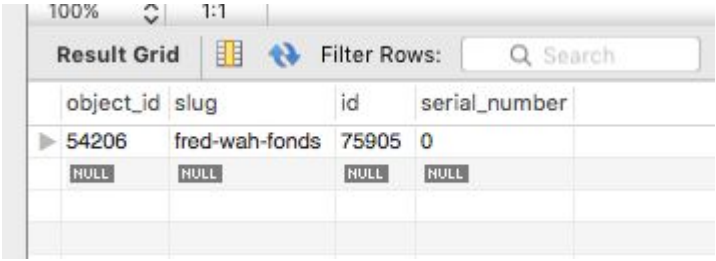
```
SELECT * FROM slug WHERE slug.slug = "fred-wah-fonds";
```

We can browse all the slugs in this table:

- `SELECT * FROM slug;`

Use slug.object_id to find the description:

```
SELECT * FROM information_object WHERE id = 54206;
```



Object Table

A short diversion...

Recalling the ORM discussion earlier:

- Most models extend ‘object’ model
- Object table row represented by object class/model
- Id’s for extended classes are derived from object class
- Ensures id’s are unique across different object types
- `SELECT * FROM object;`
- Note: class_name, id

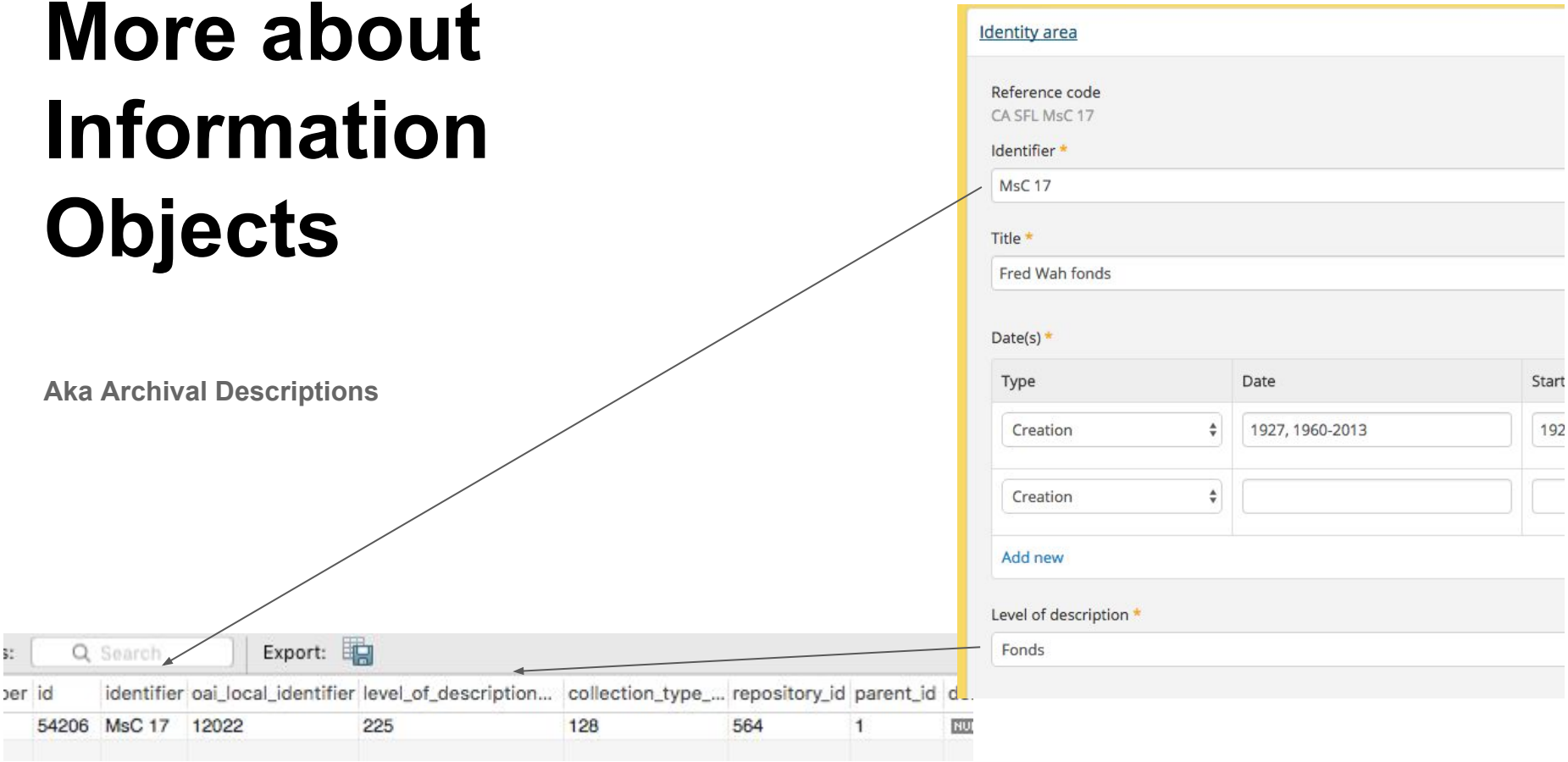
```
SELECT * FROM slug
INNER JOIN object ON object.id = slug.object_id
INNER JOIN information_object ON information_object.id = object.id
WHERE slug.slug = "fred-wah-fonds";
```

Or drop the join on object entirely:

```
SELECT * FROM slug
INNER JOIN information_object ON information_object.id = slug.object_id
WHERE slug.slug = "fred-wah-fonds";
```

More about Information Objects

Aka Archival Descriptions



I18n

Culture and translations

The i18n tables contain translated strings

- 1 to many relationship between a table and i18n equivalent
- If a translation record is not available for chosen culture
  - Display strings from default culture i18n record
- If a translation record is available for chosen culture
  - Strings will be populated from this record based on selected culture
  - If the string is null for a specific field within i18n row
    - Fall back to i18n record matching system default culture

Looking at the record for ‘fred-wah-fonds’:

```
SELECT * FROM slug
INNER JOIN information_object ON information_object.id = slug.object_id
INNER JOIN information_object_i18n
ON information_object_i18n.id = information_object.id
WHERE slug.slug = "fred-wah-fonds";
```

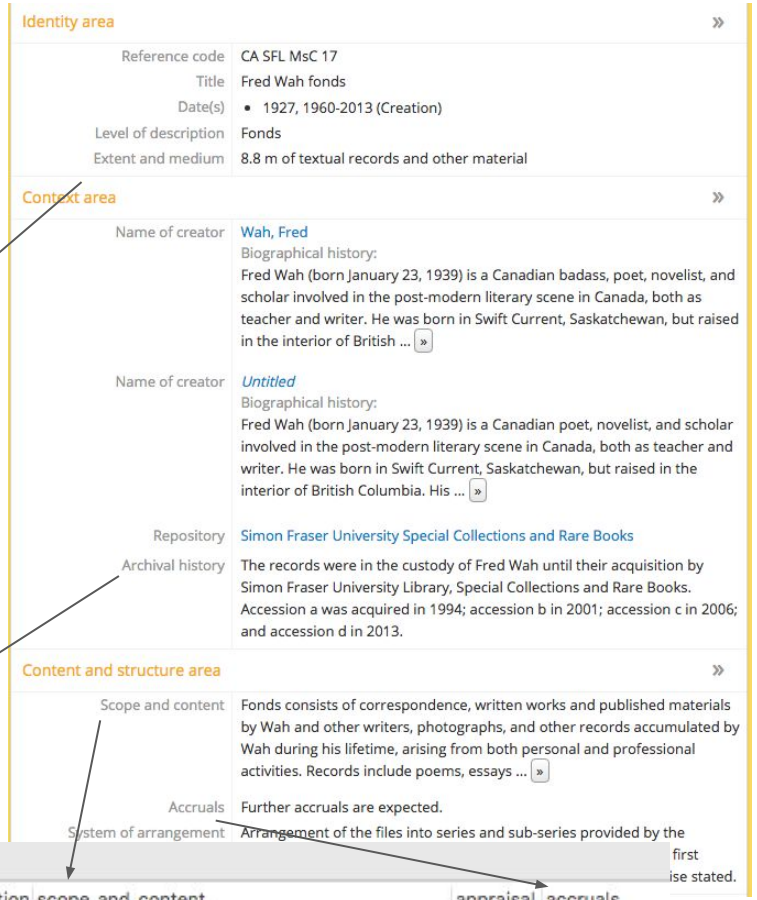
Look for ‘extent_and_medium’

Note values for field ‘culture’

I18n

i18n translatable string examples from

information_object_i18n



Result Grid | Filter Rows: Search | Export:

alternate_title	edition	extent_and_medium	archival_history	acquisition	scope_and_content	appraisal	accruals
NULL	NULL	8.8 m of textual records and other material	The records were in the custody of Fred Wah un…	NULL	Fonds consists of correspondence, written work…	NULL	Further accrual
NULL	NULL	les photos.	NULL	NULL	NULL	NULL	NULL

Events and Actors

From information_object, dates are linked to creators

- Dates → event table
- Creators → actor table

Add join from information_object to event table:

```
SELECT * FROM slug
INNER JOIN information_object ON information_object.id = slug.object_id
INNER JOIN event ON event.object_id = information_object.id
WHERE slug.slug = "example-fonds";
```

Add a join to actor table:

```
SELECT * FROM slug
INNER JOIN information_object ON information_object.id = slug.object_id
INNER JOIN event ON event.object_id = information_object.id
INNER JOIN actor ON actor.id = event.actor_id
INNER JOIN actor_i18n ON actor_i18n.id = actor.id
WHERE slug.slug = "example-fonds";
```

1. Let's add a new event (creation date) to the information object
    - A new event row is created
2. Let's add an authority record event
    - A new event record and an associated actor record created
3. Now add an authority record without dates
    - Event and actor created, but event will have null dates

Terms

Associating terms with objects

- Cross-reference table **`object_term_relation`**
- **`Example-fonds`** has an id of **`57671`**

Result Grid | Filter Rows:

id	object_id	term_id	start_date	end_date
57673	57671	443	NULL	NULL
57674	57671	445	NULL	NULL
57675	57671	447	NULL	NULL
57676	57671	449	NULL	NULL
57677	57671	451	NULL	NULL
57678	57671	453	NULL	NULL

```
SELECT * FROM object_term_relation
WHERE object_term_relation.object_id = 57671;
```

id	taxonomy_id	code	parent_id	lft	rgt	source_culture
443	35	NULL	110	616	617	en
445	35	NULL	110	618	619	en
447	42	NULL	110	620	621	en
449	42	NULL	110	622	623	en
451	78	NULL	110	624	625	en
453	78	NULL	110	626	627	en

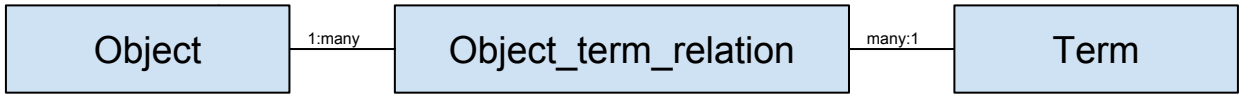
Join in the **`term`** table:

```
SELECT * FROM object_term_relation
INNER JOIN term ON term.id = object_term_relation.term_id
WHERE object_term_relation.object_id = 57671;
```

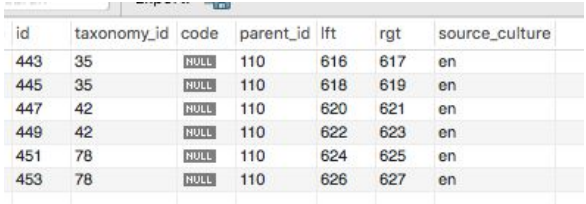
name	id	culture
Subject 1	443	en
Subject 2	445	en
Place 1	447	en
Place 2	449	en
Genre A	451	en
Genre B	453	en

Add another join to the **`term_i18n`** table:

```
SELECT * FROM slug
INNER JOIN information_object ON information_object.id = slug.object_id
INNER JOIN object_term_relation ON object_term_relation.object_id = slug.object_id
INNER JOIN term ON term.id = object_term_relation.term_id
INNER JOIN term_i18n ON term_i18n.id = term.id
WHERE slug.slug = "example-fonds";
```



Taxonomy



id	taxonomy_id	code	parent_id	lft	rgt	source_culture
443	35	NULL	110	616	617	en
445	35	NULL	110	618	619	en
447	42	NULL	110	620	621	en
449	42	NULL	110	622	623	en
451	78	NULL	110	624	625	en
453	78	NULL	110	626	627	en

name	note	id	culture
Subjects	NULL	35	en
Subjects	NULL	35	en
Places	NULL	42	en
Places	NULL	42	en
Genre	Genre terms drawn from appropriate vocabulari…	78	en
Genre	Genre terms drawn from appropriate vocabulari…	78	en

That leads us to the taxonomy table

- Each **`term`** belongs to a **`taxonomy`**
- So when we found the terms on the previous slide:

```
SELECT * FROM object_term_relation
INNER JOIN term ON term.id = object_term_relation.term_id
WHERE object_term_relation.object_id = 57671;
```

We have the taxonomy_id from the term table

Let's add the taxonomy table with a join:

```
SELECT * FROM object_term_relation
INNER JOIN term ON term.id = object_term_relation.term_id
INNER JOIN taxonomy ON taxonomy.id = term.taxonomy_id
INNER JOIN taxonomy_i18n ON taxonomy_i18n.id = taxonomy.id
WHERE object_term_relation.object_id = 57671
AND taxonomy_i18n.culture = 'en';
```

Notes and Properties

object_id	type_id	scope	user_id	source_culture	id	serial_number	content	id
57671	174	NULL	NULL	en	631	0	Example Fonds language note	631
57671	120	NULL	NULL	en	632	0	Example Fonds publication note	632
57671	125	NULL	NULL	en	633	0	Example Fonds general note	633
57671	124	NULL	NULL	en	634	0	Example Fonds archivist's notes	634

id	taxonomy_id	code	parent_id	lft	rgt	source_culture	name	id	culture
174	37	NULL	110	16	17	en	Language note	174	en

object_id	scope	name	source_culture	id	serial_number	value	id	culture
57671	NULL	language	en	420	0	a:2:{i:0;s:2:"en";i:1;s:2:"fr";}	420	en
57671	NULL	script	en	421	0	a:1:{i:0;s:4:"latn";}	421	en
57671	NULL	languageOfDescription	en	422	0	a:2:{i:0;s:2:"en";i:1;s:2:"fr";}	422	en
57671	alternativeIdentifiers	Alternative Label A	en	423	0	alt_id1	423	en

Both Notes and Properties have object_id as foreign key

Tying these records back to the objects is simply:

```
SELECT * FROM note
INNER JOIN note_i18n ON note_i18n.id = note.id
WHERE note.object_id = 57671;
```

Have a look at type_id → maps to terms table:

```
SELECT * FROM term
INNER JOIN term_i18n ON term_i18n.id = term.id
WHERE term.id = 174 AND term_i18n.culture = 'en';
```

Similarly for properties:

```
SELECT * FROM property
INNER JOIN property_i18n ON property_i18n.id = property.id
WHERE property.object_id = 57671;
```

Repository

Repository details are contained in both the repository and actor tables

- Repositories have some fields in common with actor
- Need both to get all details

id	identifier	desc_status_id	desc_detail_id	desc_identifi...	upload_limit	source_culture
57203	Example Repo 1	NULL	NULL	NULL	-1	en
NULL	NULL	NULL	NULL	NULL	NULL	NULL

```
SELECT * FROM repository
WHERE repository.id = 57203;
```

Add in the translatable strings:

geocultural_context	collecting_polici...	buildings	holdings	finding_aids	opening_times	a
Some geographical and cultural context	Record policies	NULL	NULL	NULL	M-F 9am- 5pm	

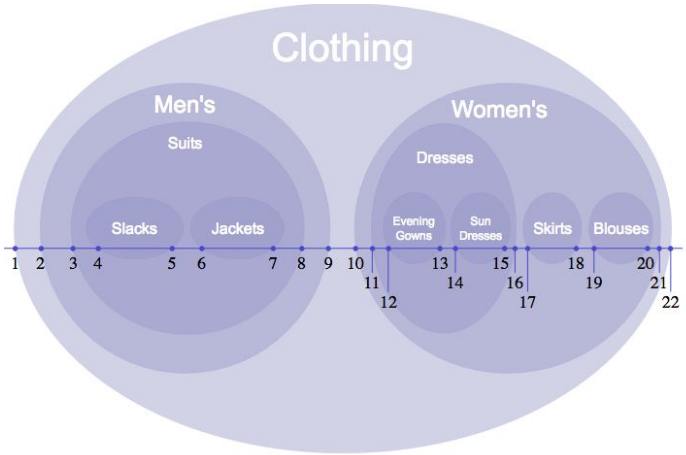
```
SELECT * FROM repository
INNER JOIN repository_i18n ON repository_i18n.id = repository.id
WHERE repository.id = 57203;
```

Add in the fields from actor & actor_i18n:

authorized_form_of_na...	dates_of_existence	history	places	legal_status	functions	mandates	internal_structur...	g
Example Repository	NULL	This is relevant history.	NULL	NULL	NULL	Mandate	Admin Structure	

```
SELECT * FROM repository
INNER JOIN repository_i18n ON repository_i18n.id = repository.id
INNER JOIN actor on actor.id = repository.id
INNER JOIN actor_i18n ON actor.id = actor_i18n.id
WHERE repository.id = 57203;
```

Nested Sets



Public domain image borrowed from https://en.wikipedia.org/wiki/Nested_set_model

What are all those lft and rgt fields?

- Nested Sets!
- A way to record hierarchical relationships among similar entities
- https://en.wikipedia.org/wiki/Nested_set_model

E.g. Information objects

- These are hierarchical objects
- Levels of Description (fonds, collection, item, part, series, etc)

Let's find a top level information_object

- Example-fonds (id: 57671, lft: 1638, rgt: 1641)
- Fred-wah-fonds (id: 54206, lft: 2, rgt: 1517)

Let's find all objects included in this object's hierarchy:

```
SELECT * FROM information_object
WHERE information_object.lft >= 1638
AND information_object.rgt <= 1641
ORDER BY information_object.lft;
```

Update Queries Caution!

- Backups!
- Know how to restore from backups!
- Practice on a backup or offline copy
- Depending on what you've done, might need to:
  - Rebuild nested sets
  - Re-index

What's next?

- Set all 'Draft' status objects to 'Published':
  - `UPDATE status SET status_id=160 WHERE type_id=158;`
- Find info object id when slug not known:
  - `SELECT id FROM information_object_i18n WHERE title='TITLEHERE';`
  - `SELECT id FROM information_object_i18n WHERE title LIKE 'TITLEHE%';`
- Get a count of descriptions in database:
  - `SELECT COUNT(*) FROM information_object_i18n;`
- Find titles containing quote characters:
  - `SELECT io.title, s.slug FROM information_object_i18n io JOIN slug s ON io.id = s.object_id WHERE io.title like '%"%';`
- See all note type terms and get a count of each:
  - ```
    SELECT term.id, term_i18n.name, COUNT(note.type_id) FROM term
    INNER JOIN term_i18n ON term.id = term_i18n.id
    INNER JOIN note ON term.id = note.type_id
    WHERE culture='en'
    GROUP BY note.type_id;
    ```

See https://www.accesstomemory.org/docs/latest/admin-manual/maintenance/cli-tools/#common-atom-database-queries

SQL Join reference: http://www.codeproject.com/Articles/33052/Visual-Representation-of-SQL-Joins
    ```


Q&A
www.accesstomemory.org
www.artefactual.com