# Advanced SQL

Pieter Joubert

March 27, 2022

Advanced SQL

# Table of Contents

1. Associative Entities

2. Joins

3. Lecture summary

# Week 9 Information

### NoSQL/GraphDB

Please note that the second session this week will focus on NoSQL/GraphDB, and will be given by another lecturer. Keep an eye on Canvas as I might upload some additional SQL videos there.

# New Database

## Backup and Import

To be able to all start from the same page I've made a script available on Canvas that will import a clean and up-to-date version of the *todo_list* database. The two commands below (run in the terminal or shell) export a database to a .sql file, and then import the same file into a database. Please note that the database must be created *first* when importing.

```
pg_dump todolist > ~joubertp/Documents/todo_list.sql
psql todolist < ~joubertp/Documents/todo_list.sql
```

Section 1

# Associative Entities

# Auto Increment PRIMARY KEYS

- We have been manually inserting primary keys into our database.
- While this works, it is something that is prone to error, and something would be better to automate.
- The example below is use to make the *todo_item_id* into an auto-incremented primary key.

```
todo_item_id integer GENERATED ALWAYS AS IDENTITY
    NOT NULL
```

- Note that this example would be best used when creating the table (and as such is included in the import script)

# Associative Entities

- So far we have only *one-to-many* relationships in our database.
- For example, one *owner* can own many *todo_items*.
- As our database gets more complex we are also going to need to implement more complex relationships.
- One example of such a relationship could be a *many-to-many* relationship between *owners* and *projects*.
- We will now need to add a new table linking *owner* and *project*.

# Associative Entity SQL

- The code to create a new Associative Entity is shown below:

```sql
CREATE TABLE owner_project (
      owner_id integer NOT NULL,
      project_id integer NOT NULL,
      PRIMARY KEY(owner_id, project_id)
);
```

- Note that we do not use auto-incremented values for the primary keys, and that we have a *compound* primary key made up of two fields.
- We can now insert some rows that link an owner to a project.

Section 2

# Joins

# Joins

- Now that we have all of these FK/PK links set up in our database we need to start using these relationships.
- What we essentially need to do is *join* together two or more tables, based on the links that we have created.
- The SQL here can get a little complex but this is where *set theory* actually becomes useful, as we essentially performing a UNION, INTERSECTION or COMPLEMENT on two or more sets (two or more tables).
- If you are unsure which kind of *join* to use, consider drawing out the relationships as a Venn diagram.

# INNER JOIN

- An INNER JOIN selects rows that match in both tables.
- For example we might join *owner* and *todo_item* to get the owner name associated with a todo_item, instead of the owner_id:

```
SELECT todo_item.description, owner.owner_name,
    owner.owner_surname
FROM todo_item
INNER JOIN owner ON todo_item.owner_id=owner.
    owner_id;
```

# Result

```
     description     | owner_name |  owner_surname
---------------------+------------+-----------------
 Align telescope     | Carl       | Sagan
 Align telescope     | Neil       | de Grasse Tyson
 Write Report        | Neil       | de Grasse Tyson
 Send Report         | Neil       | de Grasse Tyson
 Send Another Report | Neil       | de Grasse Tyson
 Check Email         | Neil       | de Grasse Tyson
 Check Email         | Andrea     | Ghaez
 Align telescope     | Carl       | Sagan
 Check telescope     | Andrea     | Ghaez
```

# INNER JOIN with all columns

- Expanding on the previous example to include all the Foreign Key Columns

```sql
SELECT
    priority.priority_name, todo_item.description, owner.owner_name, owner.owner_surname, project.project_name, status.status_name, context.context_name, due_date
    FROM todo_item
    INNER JOIN owner ON todo_item.owner_id = owner.owner_id
    INNER JOIN project ON todo_item.project_id = project.project_id
    INNER JOIN priority ON todo_item.priority_id = priority.priority_id
    INNER JOIN context ON todo_item.context_id = context.context_id
    INNER JOIN status ON todo_item.status_id = status.status_id;
```

# LEFT/RIGHT JOIN

- In our example let's add one more owner to our owner table:

```sql
INSERT INTO owner (owner_name, owner_surname)
    VALUES ('Thomas','Kepler');
```

- Note that for now we don't link this new owner to a todo_item.

# RIGHT JOIN

- The example below performs a *RIGHT* join between our todo_item and owner tables.

```
SELECT todo_item.description, owner.owner_name,
    owner.owner_surname
FROM todo_item
RIGHT JOIN owner ON todo_item.owner_id=owner.
    owner_id;
```

- Note how we now have additional owner, that is not linked to a todo_item.

# FULL JOIN

- The example below performs a *FULL* join between our todo_item and owner tables.

```
SELECT todo_item.description, owner.owner_name,
    owner.owner_surname
FROM todo_item
FULL JOIN owner ON todo_item.owner_id=owner.
    owner_id;
```

- In this example our results still looks the same. Remember that our todo_item has a foreign key relationship with the owner table, so a todo_item *must* have an owner and our *FULL* join will not have an additional todo_item entry.

# JOINs

- We've just looked at some basic examples of JOINs.
- The correct type of JOIN to use depends on the database structure you've design.
- Most often an *INNER JOIN* will solve your problem, but keep in mind that there are other options when performing a JOIN.
- And finally, we can combine any of the other functions and SQL code with a JOIN to create more complex and focused queries.

Section 3

## Lecture summary

# Lecture summary

- Associative Entities
- JOINs

# Questions

**Thank you! Questions?**