Databases

BCS1510

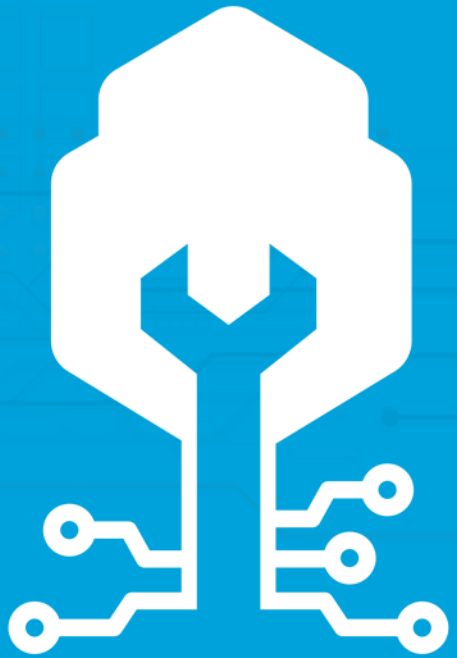
Dr. Katharina Schneider &
Dr. Tony Garnock-Jones*

🗓️ Week 2 - Lecture 1

📍 EPD150 MSM Conference Hall

Adapted with gratitude from the original lectures by Dr. Ashish Sai

*CC both of us when emailing please!



Maastricht University

Before we start

- Reminder: put yourself in a group on Canvas

What have we looked at so far?

- Database
  - Collection of logically related data
  - Two formal definitions
- Database Management System
  - Program used for defining and maintaining this database.
  - DBMS Evolution
  - From file systems to blockchains and big data.
- Data Models
  - Conceptual and Physical Data Models
- Relational Data Model
  - An overview of entity and relationship model
- Relational Algebra
  - Introduction to Relational Algebra.
- SQL as data definition language
  - Creating and altering tables

Open Question from last lecture

- Can we update a value only in a tuple?

```
UPDATE table name
SET column1 = value1, column2 = value2, …
WHERE condition;
```

- Be careful: Where clause is not necessary. But if you omit it, all records in the table will be updated

Example UPDATE

- Set the firstname of Ann in table customer to Anne

```
UPDATE customers
SET firstname = 'Ann'
WHERE firstname = 'Anne';
```

Learning Objectives

- Understand SQL Fundamentals
  - Learn the basic functions of SQL as a data manipulation language, including the structure and execution of queries using SELECT, FROM, and WHERE clauses.
- Query Formulation and Execution
  - Develop skills in crafting effective SQL queries to manipulate and retrieve data, understanding how to apply conditions, expressions, and patterns to refine results
- Advanced Query Techniques
  - Gain proficiency in handling complex SQL operations such as multi-relation queries, joins, and managing NULL values with an understanding of SQL's three-valued logic
- Practical Application of SQL
  - Apply SQL knowledge to real-world database scenarios, using different types of joins and advanced query syntax to solve practical problems and optimize database interactions.

SQL (Part 2)

SQL as Data Manipulation Language

Select

SELECT-FROM-WHERE Statements

```
SELECT desired attributes
FROM one or more tables
WHERE condition about tuples of the tables;
```

- The where clause is optional

SELECT is projection (projecting rows/attributes)
FROM relation (choose a table)
WHERE is selection (row-related condition)

Example

- Using relation Customers: find the last names of the customers who live in Limerick

```
SELECT lastname
FROM customers
WHERE city = 'Limerick';
```

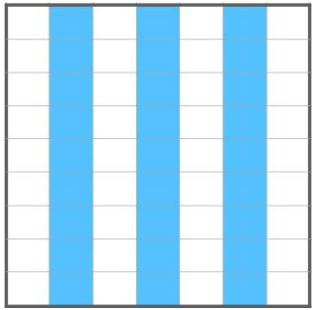
- The answer is a relation with a single attribute – lastname – and tuples (rows) with the lastname of each customer in Limerick

Lastname
Jones
Doe

Formal Semantics: Single-Relation Query

- Begin with the relation in the FROM clause
- Apply the *selection* indicated by the WHERE clause
- Apply the *projection* indicated by the SELECT clause

Selection



Projection

Think about your tutorial

- Q1: Find the name of all employees (i.e., persons) who work for the City Bank company

- Start with the relation (from FROM clause)
- -> $(occupation)$

- Apply the selection (from WHERE clause)
- -> $(\sigma_{company_{name}='City\ Bank'}(occupation))$

- Apply the projection (from SELECT clause)
- -> $\pi_{person_{name}}(\sigma_{company_{name}='City\ Bank'}(occupation))$

Operational Semantics: Single-Relation Query

- Think of a tuple variable visiting each tuple of the relation mentioned in FROM
- Check if the “current” tuple satisfies the WHERE clause
- If so, compute the attributes or expressions of the SELECT clause using the components of this tuple

Select * Clause

- When there is one relation in the FROM clause, * in the SELECT clause stands for “all attributes of this relation”
- Example

```
SELECT *
FROM customers
WHERE city = 'Limerick';
```

customer_id	firstname	lastname	city	address	email
9999999999	Norah	Jones	Limerick	2 Thomas St.	nj@yahoo.com
1231231231	John	Doe	Limerick	NULL	NULL

Renaming attributes

- If you want the results to have different attribute names, use AS to rename an attribute
- Example

```
SELECT lastname AS surname
FROM customers
WHERE city = 'Limerick';
```

surname
Jones
Doe

Expressions in SELECT Clauses

- Any expression that makes sense can appear as an element of a SELECT clause
- Example

```
SELECT model,price,price*1.4 AS priceUSD
FROM PCs
```

model	price	priceUSD
1001	2114.00	2959.60
1002	995.00	1393.00
1003	478.00	669.20
1004	649.00	908.60
1005	630.00	882.00
1006	1049.00	1468.60
1007	510.00	714.00
1008	770.00	1078.00
1009	650.00	910.00
1010	770.00	1078.00
1011	959.00	1342.60
1012	649.00	908.60
1013	529.00	740.60

Constants as Expressions in SELECT Clause

```
SELECT model,price*1.4 AS price, 'USD' AS currency
FROM PCs
```

model	price	currency
1001	2959.60	USD
1002	1393.00	USD
1003	669.20	USD
1004	908.60	USD
1005	882.00	USD
1006	1468.60	USD
1007	714.00	USD
1008	1078.00	USD
1009	910.00	USD
1010	1078.00	USD
1011	1342.60	USD
1012	908.60	USD
1013	740.60	USD

IS NULL

```
SELECT customer_id, lastname
FROM customers
WHERE email = NULL;
```

Customer_id	lastname

```
SELECT customer_id, lastname
FROM customers
WHERE email IS NULL;
```

Customer_id	Lastname
1231231231	Doe
1234567890	Murphy

Complex Conditions in WHERE Clause

- Boolean operators AND, OR, NOT
- Comparisons =, <>, <, >, <=, >=
- And many other operators that produce Boolean-valued results

Example: Complex Conditions in WHERE Clause

- Using relation PCs , find the models with speed greater than or equal to 2.00 and price less than 1000.00

```
SELECT model
FROM PCs
WHERE speed >= 2.0 AND price < 1000.0;
```

model
1002
1004
1005
1007
1008
1009
1010
1012
1013

Patterns in WHERE Clause

- A condition can compare a string to a pattern by
  - <Attribute> LIKE <pattern>

  or
  - <Attribute> NOT LIKE <pattern>

- Pattern is a quoted string

- Wildcards in LIKE
  - %: matches any sequence of characters (string)
  - _: matches exactly one character

Example 1: LIKE in WHERE clause

- Using relation Customers, find the names of all customers who live in Dublin and have ‘Jervis St.’ in their address

```
SELECT firstname, lastname
FROM customers
WHERE city =  'Dublin' AND address LIKE '%Jervis St.%';
```

Example 2: LIKE in WHERE clause

- Using relation Sales, find all details about sales made in December (any year)

```
SELECT *
FROM Sales
WHERE day LIKE '____-12%';
```

- Watch out, use exactly 4 times underline sign!! (as the year has 4 characters)

NULL Values

- Tuples in SQL relations can have NULL as a value for one or more components

- Meaning depends on context. Two common cases:
  - Missing value: e.g., we know a customer has some address, but we do not know what it is
  - Inapplicable: e.g., the value of attribute name of spouse for an unmarried person

Three-Valued Logic

- The logic of conditions in SQL is really 3-valued logic: TRUE, FALSE, UNKNOWN
- Comparing any value (including NULL itself) with NULL yields UNKNOWN
- A tuple is in a query answer iff the WHERE clause is TRUE (not FALSE or UNKOWN)

Three-Valued Logic - Overview

T = TRUE, F = FALSE, U = UNKNOWN

OR	*T*	*F*	*U*
T	T	T	T
F	T	F	U
U	T	U	U

AND	*T*	*F*	*U*
T	T	F	U
F	F	F	F
U	U	F	U

	NOT
T	F
F	T
U	U

Three-Valued Logic – NULL in Comparisons

Id	Name	Department	Salary	bonus
1	Alice	HR	50000	5000
2	Bob	IT	NULL	4000
3	Charlie	Sales	60000	NULL
4	David	HR	NULL	NULL

```
SELECT *
FROM Employees
WHERE salary > 40000;
```

- Result contains only Alice and Charlie
- WHERE filters out UNKOWN and FALSE

Id	Name	Department	Salary	bonus
1	Alice	HR	50000	5000
3	Charlie	Sales	60000	NULL

Three-Valued Logic – AND operator

Id	Name	Department	Salary	bonus
1	Alice	HR	50000	5000
2	Bob	IT	NULL	4000
3	Charlie	Sales	60000	NULL
4	David	HR	NULL	NULL

```
SELECT *
FROM Employees
WHERE salary > 40000 AND bonus > 3000;
```

- Alice (50000 > 40000 AND 5000> 3000 -> TRUE AND TRUE -> TRUE
- Bob (NULL > 40000 AND 4000> 3000 -> UNKNOWN AND TRUE -> UNKNOWN
- Charlie (60000 > 40000 AND NULL > 3000 -> TRUE AND UNKNOWN -> UNKNOWN
- David (NULL > 40000 AND NULL > 3000 -> UNKNOWN AND UNKNOWN -> UNKNOWN

Id	Name	Department	Salary	bonus
1	Alice	HR	50000	5000

Three-Valued Logic – OR operator

Id	Name	Department	Salary	bonus
1	Alice	HR	50000	5000
2	Bob	IT	NULL	4000
3	Charlie	Sales	60000	NULL
4	David	HR	NULL	NULL

```
SELECT *
FROM Employees
WHERE salary > 40000 OR bonus > 3000;
```

- Alice (50000 > 40000 OR 5000> 3000 -> TRUE OR TRUE -> TRUE
- Bob (NULL > 40000 OR 4000> 3000 -> UNKNOWN OR TRUE -> TRUE
- Charlie (60000 > 40000 OR NULL > 3000 -> TRUE OR UNKNOWN -> TRUE
- David (NULL > 40000 OR NULL > 3000 -> UNKNOWN OR UNKNOWN -> UNKNOWN

Id	Name	Department	Salary	bonus
1	Alice	HR	50000	5000
2	Bob	IT	NULL	4000
3	Charlie	Sales	60000	NULL

Three-Valued Logic - Laws

- 2-valued laws != 3-valued laws

- Some common laws hold in 3-valued logic
  - Example: commutativity of AND, i.e., p AND q is the same as q AND p

- But not others, e.g. the law of the excluded middle:
  - Example: p OR NOT p = TRUE
  - When p = UNKNOWN, the left side is UNKOWN!!!

Short Break

- **10** minutes

Multi-relation Query

- Interesting queries often combine data from more than one relation (table)
- We can address several relations in one query by listing them all in the FROM clause
- Distinguish attributes of the same name by <relation>.<attribute>

Example: Multi-relation Query

- Query: Find the names and the addresses of all customers from Limerick who have made a purchase on the 20th of December 2013
- Relations to use:
  - Customers(customer_id, firstname, lastname, city, address, email)
  - Sales (customer_id, model, quantity, day, paid, type_of_payment)

```
SELECT firstname, lastname, address
FROM customers, sales
WHERE customers.customer_id = sales.customer_id AND
    city = 'Limerick' AND day = '2013-12-20' ;
```

DISTINCT

```
SELECT firstname, lastname, address
FROM customers, sales
WHERE customers.customer_id = sales.customer_id AND
    city = 'Limerick' AND day = '2013-12-20' ;
```

- Results in

firstname	lastname	address
Norah	Jones	2 Thomas St.
Norah	Jones	2 Thomas St.

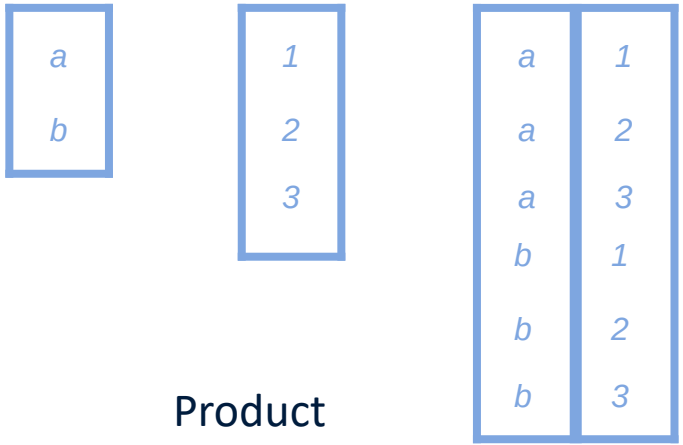
- Because the customer has bought 2 models on this date

```
SELECT DISTINCT firstname, lastname, address
FROM customers, sales
WHERE customers.customer_id = sales.customer_id AND
    city = 'Limerick' AND day = '2013-12-20' ;
```

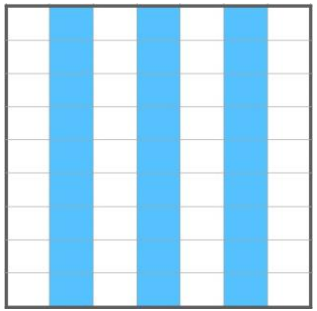
firstname	lastname	address
Norah	Jones	2 Thomas St.

Formal Semantics: Multi-relation Query

- Almost the same as for single-relation queries
  - Start with the *product of all the relations* in the FROM clause
  - Apply the *selection* condition from the WHERE clause
  - *Project* onto the list of attributes and expressions in the SELECT clause



Selection



Projection

Operational Semantics: Multi-relation Query

- Imagine one tuple-variable for each relation in the FROM clause
- These tuple-variables visit each combination of tuples, one from each relation
- If the tuple-variables are pointing to tuples that satisfy the WHERE clause, send these tuples to the SELECT clause

Multi-Relation Query: Alternative (better) Syntax

```
SELECT firstname, lastname, address
FROM customers JOIN sales ON customers.customer_id = sales.customer_id
WHERE city = 'Limerick' AND day = '2013-12-20' ;
```

- It might be better to use JOIN … ON instead of the comma operator because it provides more information to the DBMS about what we intend to do. Thus, the DBMS may optimize the query better (in general)

USING

- Shorthand way to specify a join condition when both tables share a column with the same name

- It simplifies queries by avoiding ON conditions

```
SELECT firstname, lastname, address
FROM customers JOIN sales ON customers.customer_id = sales.customer_id
WHERE city = 'Limerick' AND day = '2013-12-20' ;
```

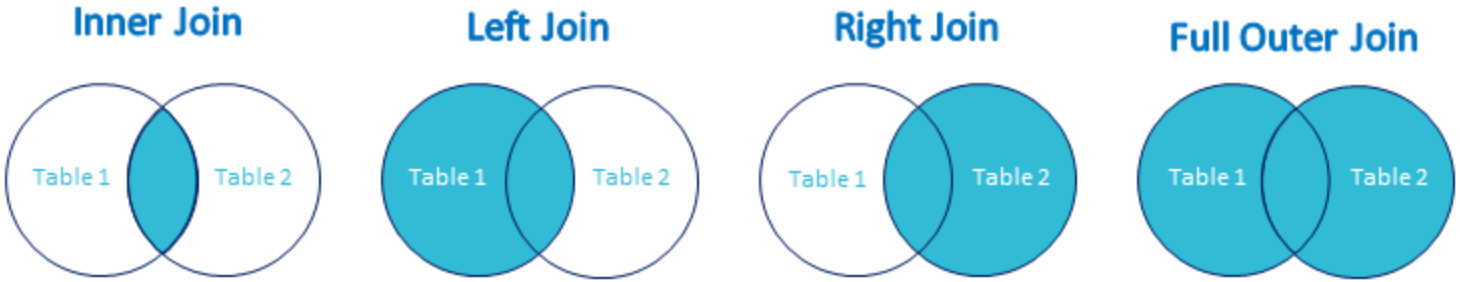
```
SELECT firstname, lastname, address
FROM customers JOIN sales USING (customer_id)
WHERE city = 'Limerick' AND day = '2013-12-20' ;
```

Different Types of JOIN

- (INNER) JOIN
  - Combines matching* rows from two tables
- LEFT (OUTER) JOIN
  - Returns all rows from the left table along with the matching* rows from the right table
  - Similar to JOIN, but if a row in the left table has not match in the right table, then it is matched with a row containing NULL values
- Right (OUTER) JOIN
  - Returns all rows from the right table along with the matching* rows from the left table
  - Same as LEFT JOIN with switched tables
- FULL (OUTER) JOIN
  - Returns all matching* rows in one of the tables
  - Union of the result from LEFT JOIN and RIGHT JOIN
- CROSS JOIN
  - Produces a Cartesian product of the two tables. Without any condition. No matching.
- NATURAL JOIN
  - Automatically joins tables based on columns with the same name and compatible data types in both tables

*with matching, we refer to a match based on the condition provided

JOINs (Visualization)



INNER JOIN - EXAMPLE

```
SELECT students.student_id, students.name,
      courses.course_name
FROM students
INNER JOIN courses
ON students.course_id = courses.course_id;
```

Student_id	name	course_name
1	Alice	Math
2	Bob	Physics
4	David	Biology

students

id	name	Course_id
1	Alice	101
2	Bob	102
3	Charlie	NULL
4	David	104

courses

Course_id	Course_name
101	Math
102	Physics
103	Chemistry
104	Biology

LEFT JOIN - EXAMPLE

```
SELECT students.student_id, students.name,
       courses.course_name
FROM students
LEFT JOIN courses
ON students.course_id = courses.course_id;
```

Student_id	name	course_name
1	Alice	Math
2	Bob	Physics
3	Charlie	NULL
4	David	Biology

students

id	name	Course_id
1	Alice	101
2	Bob	102
3	Charlie	NULL
4	David	104

courses

Course_id	Course_name
101	Math
102	Physics
103	Chemistry
104	Biology

RIGHT JOIN - EXAMPLE

```
SELECT students.student_id, students.name,
       courses.course_name
FROM students
RIGHT JOIN courses
ON students.course_id = courses.course_id;
```

Student_id	name	course_name
1	Alice	Math
2	Bob	Physics
NULL	NULL	Chemistry
4	David	Biology

students

id	name	Course_id
1	Alice	101
2	Bob	102
3	Charlie	NULL
4	David	104

courses

Course_id	Course_name
101	Math
102	Physics
103	Chemistry
104	Biology

FULL JOIN - EXAMPLE

```
SELECT students.student_id, students.name,
       courses.course_name
FROM students
FULL JOIN courses
ON students.course_id = courses.course_id;
```

Student_id	name	course_name
1	Alice	Math
1	Alice	Physics
1	Alice	Chemistry
1	Alice	Biology
2	Bob	Math
2	Bob	Physics
2	Bob	Chemistry
2	Bob	Biology
3	Charlie	Math
3	Charlie	Physics
…	…	…

students

id	name	Course_id
1	Alice	101
2	Bob	102
3	Charlie	NULL
4	David	104

courses

Course_id	Course_name
101	Math
102	Physics
103	Chemistry
104	Biology

FULL JOIN - EXAMPLE

```
SELECT students.student_id, students.name,
       courses.course_name
FROM students
FULL JOIN courses
ON students.course_id = courses.course_id;
```

DOES NOT WORK WITH MYSQL

students

id	name	Course_id
1	Alice	101
2	Bob	102
3	Charlie	NULL
4	David	104

courses

Course_id	Course_name
101	Math
102	Physics
103	Chemistry
104	Biology

FULL JOIN - Alternative

```
SELECT students.student_id, students.name,
       courses.course_name
FROM students
LEFT JOIN courses
ON students.course_id = courses.course_id

UNION

SELECT students.student_id, students.name,
       courses.course_name
FROM students
RIGHT JOIN courses
ON students.course_id = courses.course_id;
```

students

id	name	Course_id
1	Alice	101
2	Bob	102
3	Charlie	NULL
4	David	104

courses

Course_id	Course_name
101	Math
102	Physics
103	Chemistry
104	Biology

NATURAL JOIN - EXAMPLE

```
SELECT students.student_id, students.name,
       courses.course_name
FROM students
NATURAL JOIN courses;
```

Student_id	name	course_name
1	Alice	Math
2	Bob	Physics
4	David	Biology

students

id	name	Course_id
1	Alice	101
2	Bob	102
3	Charlie	NULL
4	David	104

courses

Course_id	Course_name
101	Math
102	Physics
103	Chemistry
104	Biology

NATURAL JOIN

- Assume: both tables contain attributes with names $a_1, a_2, \ldots, a_n$
- A tuple from left table is combined with a tuple from right table if both tuples have the same values for attributes $a_1, a_2, \ldots, a_n$
- Only one copy of each of the attributes $a_1, a_2, \ldots, a_n$ is present in the result

NATURAL JOIN – Example

- Consider the DISTINCT example:

```
SELECT DISTINCT firstname, lastname, address
 FROM customers, sales
 WHERE customers.customer_id = sales.customer_id AND
       city = 'Limerick' AND day = '2013-12-20' ;
```

- Alternative:

```
SELECT DISTINCT firstname, lastname, address
FROM customers NATURAL JOIN sales
WHERE city = 'Limerick' AND day = '2013-12-20' ;
```

- Q: What will be the difference between the results of the two queries above?

NATURAL JOIN – Example (continued)

```
SELECT DISTINCT firstname, lastname, address
FROM customers, sales
WHERE customers.customer_id = sales.customer_id AND
      city = 'Limerick' AND day = '2013-12-20' ;
```

firstname	lastname	address
Norah	Jones	2 Thomas St.

```
SELECT DISTINCT firstname, lastname, address
FROM customers NATURAL JOIN sales
WHERE city = 'Limerick' AND day = '2013-12-20' ;
```

firstname	lastname	address
Norah	Jones	2 Thomas St.

- Only works as an alternative because the attribute we join the two tables on – customer_id – is called the same in both tables

Difference INNER JOIN – NATURAL JOIN

```
SELECT students.student_id, students.name,
       courses.course_name
FROM students
INNER JOIN courses
ON students.course_id = courses.course_id;
```

- Gave the same output as

```
SELECT students.student_id, students.name,
       courses.course_name
FROM students
NATURAL JOIN courses;
```

So? What is the difference?

```
SELECT *
FROM students
INNER JOIN courses
ON students.course_id = courses.course_id;
```

Student_id	name	Course_id	Course_id	course_name
1	Alice	101	101	Math
2	Bob	102	102	Physics
4	David	104	104	Biology

```
SELECT *
FROM students
NATURAL JOIN courses;
```

Course_id	Student_id	name	course_name
101	1	Alice	Math
102	2	Bob	Physics
104	4	David	Biology

Explicit Tuple-Variables

- Sometimes, a query needs to use two copies of the same relation (self-join) (e.g., when rows in a table have relationships with other rows in the same table)
- Distinguish copies by following the relation name by the name of a tuple-variable in the FROM clause
- It is always an option to rename relations this way even when not essential

Tuple-Variables – Example (1)

- From the PC Shop customers, find all pairs of customer_id which belong to customers from the same city
  - Do not produce pairs like (9999999999, 9999999999)
  - Produce pairs in alphabetic order (1231231231, 9999999999) instead of (9999999999, 1231231231)

Customers

customer_id	firstname	lastname	city	address	email
9999999999	Norah	Jones	Limerick	2 Thomas St.	nj@yahoo.com
1234567890	Paul	Murphy	Cork	20 O'Connell St.	NULL
1122334455	Ann	O'Brien	Dublin	1 Jervis St.	aob@ul.ie
1231231231	John	Doe	Limerick	NULL	NULL
9876543210	Jack	Murphy	Galway	101 O'Connell St.	jm@ul.ie

```
SELECT c1.customer_id, c2.customer_id
FROM Customers c1, Customers c2
WHERE c1.city = c2.city AND c1.customer_id < c2.customer_id;
```

customer_id	customer_id
1231231231	9999999999

Tuple-Variables – Example (2)

employee_id	name	manager-_id
1	Alice	NULL
2	Bob	1
3	Charlie	1
4	David	2

managers

```
SELECT e.name AS Employee, m.name AS Manager
FROM managers e, managers m
WHERE e.manager_id = m.employee_id;
```

Employee	Manager
Bob	Alice
Charlie	Alice
David	Bob

References about Joins

- https://dev.mysql.com/doc/refman/8.4/en/join.html
- https://www.w3schools.com/sql/sql_join.asp
- https://docs.oracle.com/cd/B28359_01/server.111/b28286/queries006.htm#SQLRF52333

- Also check the references at the bottom of the Wikipedia article: https://en.wikipedia.org/wiki/Join_(SQL)

In Class Exercises

- Q1: List all customers that live in Limerick
- Q2: Retrieve all customer ID and full name of customers who have made at least one purchase
- Q3: List all customers who have never made a purchase

Contact details

- If you have any issues throughout the course, contact us via email:
- Tony: tony.garnock-jones@maastrichtuniversity.nl
- Katharina: k.schneider@maastrichtuniversity.nl
- Or send a message on Discord
- (https://discord.gg/Be2KSF8QG6)

Questions?

See you tomorrow