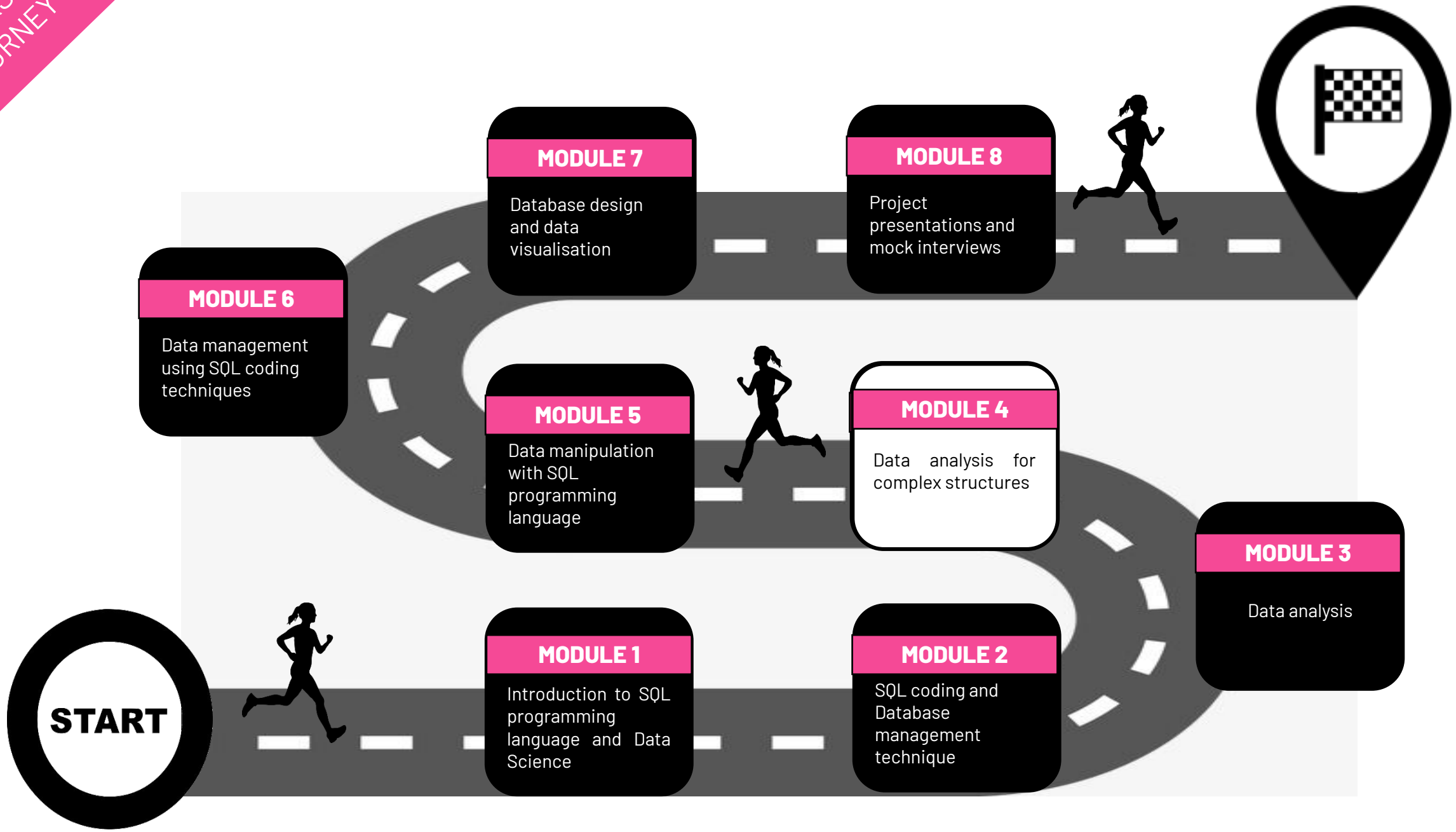
WELCOME TO CFG

YOUR INTRODUCTION TO DATABASES & SQL PROGRAMMING LANGUAGE

CODE FIRST GIRLS

TECH OPENS UP LIMITLESS OPPORTUNITIES FOR GIRLS

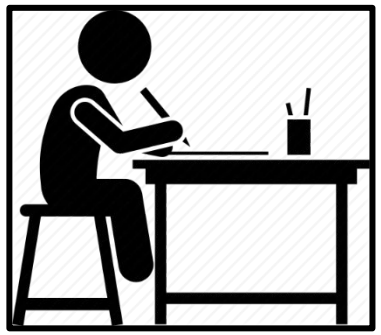
COURSE JOURNEY



04 DATA ANALYSIS FOR COMPLEX STRUCTURES

- **Joins – create complex data structures within DB**
- **Data subset unions**
- **SQL programming – subqueries**

HOMEWORK REVIEW



- The main objective of week 3 homework was to write various queries using logical operators, keywords, aggregation and sorting techniques to analyse sales data for a chain of stores.

- Let's walk through and review the correct answers together.

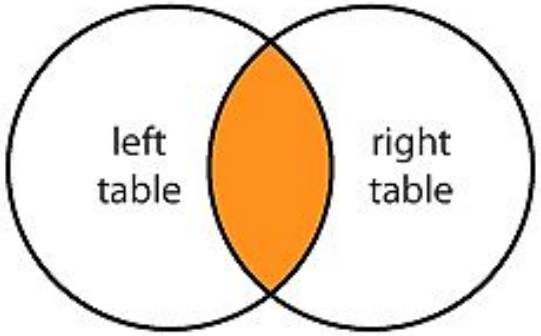
JOIN

- SQL JOIN combines columns from two or more tables, based on a related column between them, in a single result set.

- There are many different types of JOINs:

- **Inner Join**
- **Outer Join**
  - **Left Outer Join**
  - **Right Outer Join**
  - **Full Outer Join**
- **Self and Equi Join**
- **Cross Join**
- **Natural Join**
- **Straight Join**
- **Keyword Join**

INNER JOIN



- Most typical JOIN, it joins column in first table to second table
- It returns rows when there is at least one match in both tables
- Cannot deal with NULL values

Table 1	
ID	VALUE
1	First
2	Second
3	Third
4	Fourth
5	Fifth

Table 2	
ID	VALUE
1	First
2	Second
3	Third
6	Sixth
7	Seventh

SYNTAX

```
SELECT
<alias1>.<column_name>,
<alias1>.<column_name>,
<alias2>.<column_name>

FROM <table_name1>  <alias1>
INNER JOIN
<table_name2>  <alias2>
ON
<alias1>.<column_name> =
<alias2>.<column_name>;
```

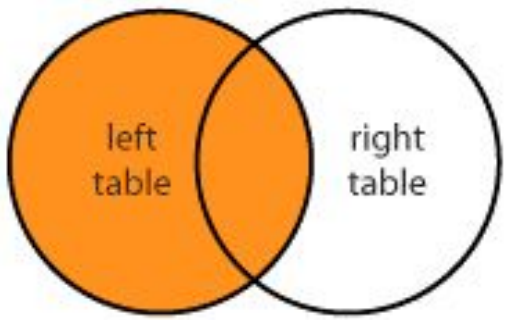
EXAMPLE

What are our customers' email addresses?

```
SELECT
c.name, c.surname,
e.email_address
FROM customers c
INNER JOIN
email_address e
ON
c.customer_id =
e.email_address_customer_id;
```

- **INNER JOIN**
- **ON clause**

LEFT OUTER* JOIN



- All rows from the left side with the matching rows from the right side will be returned

- If there are no columns matching in the right table, it returns NULL values

Table 1	
ID	VALUE
1	First
2	Second
3	Third
4	Fourth
5	Fifth

Table 2	
ID	VALUE
1	First
2	Second
3	Third
6	Sixth
7	Seventh

The '**OUTER**' keyword is optional

SYNTAX

```
SELECT
<alias1>.<column_name>,
<alias1>.<column_name>,
<alias2>.<column_name>

FROM <table_name1>  <alias1>
LEFT JOIN
<table_name2>  <alias2>
ON
<alias1>.<column_name> =
<alias2>.<column_name>;
```

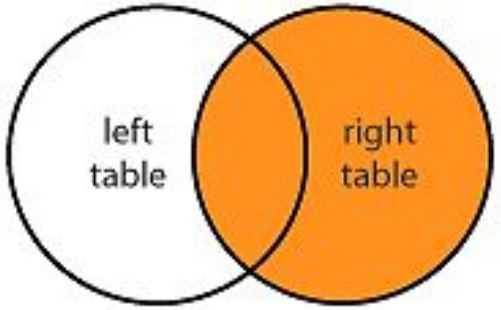
EXAMPLE

What are our customers' email addresses including those that we don't have an email address for?

```
SELECT
c.name, c.surname,
e.email_address
FROM customers c
LEFT JOIN
email_address e
ON
c.customer_id =
e.email_address_customer_id;
```

- **LEFT OUTER JOIN**
- **ON clause**

RIGHT OUTER* JOIN



- Opposite of the LEFT OUTER JOIN, i.e. all rows from the right side with the matching rows from the left side will be returned
- If there are no columns matching in the left table, it returns NULL values

Table 1	
ID	VALUE
1	First
2	Second
3	Third
4	Fourth
5	Fifth

Table 2	
ID	VALUE
1	First
2	Second
3	Third
6	Sixth
7	Seventh

The '**OUTER**' keyword is optional

SYNTAX

```
SELECT
<alias1>.<column_name>,
<alias1>.<column_name>,
<alias2>.<column_name>

FROM <table_name1>  <alias1>
RIGHT JOIN
<table_name2>  <alias2>
ON
<alias1>.<column_name> =
<alias2>.<column_name>;
```

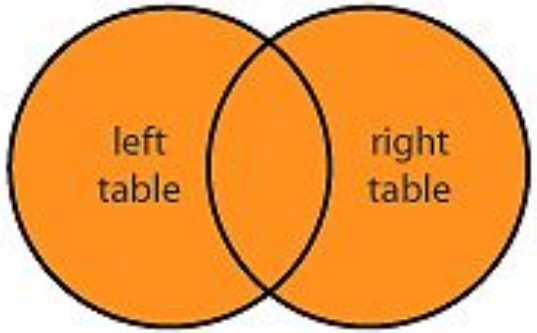
EXAMPLE

What are the email addresses we have, including those email that we don't have a matching customer for?

```
SELECT
c.name, c.surname,
e.email_address
FROM customers c
RIGHT JOIN
email_address e
ON
c.customer_id =
e.email_address_customer_id;
```

- **RIGHT OUTER JOIN**
- **ON clause**

FULL OUTER* JOIN



The '**OUTER**' keyword is optional

- Combines left outer join and right outer join
- Returns rows from either table when the conditions are met and returns a null value when there is no match
- MySQL DOES NOT SUPPORT FULL OUTER JOIN SYNTAX!
- But we can simulate this join by combining LEFT and RIGHT joins with UNION (don't worry about it now – we learn more about UNIONs soon!

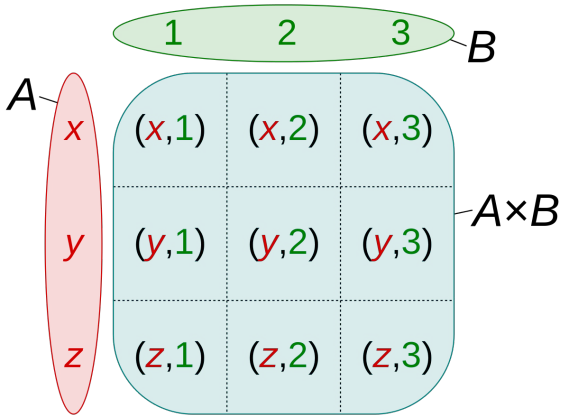
Table 1	
ID	VALUE
1	First
2	Second
3	Third
4	Fourth
5	Fifth

Table 2	
ID	VALUE
1	First
2	Second
3	Third
6	Sixth
7	Seventh

PRACTICE

- We are going to create two simple tables and join them together using different types of joins.
- The most important thing for us right now is to understand the join mechanism and the SQL syntax that we need to write to perform a join.
- Let's create two tables aka two fruit baskets and see how we can join them!
- Type along with your instructor ☺️.

CROSS JOIN



- Simplest, but quite inefficient
- There is no WHERE clause nor any condition for a join, hence all rows from both tables are used
- It returns a Cartesian Product – multiples of the record number of both tables

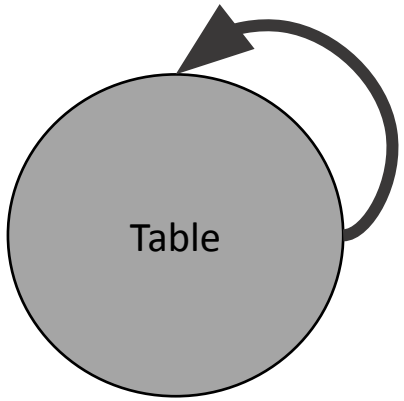
Table 1

ID	VALUE
1	First
2	Second
3	Third

Table 2

ID	VALUE
1	First
2	Second
3	Third
4	Fourth

SELF JOIN



- You can JOIN the table on itself! Sometimes it can be useful.
- No special syntax, but we must alias the table used with different aliases.
- We will have the same table on left and right side of this join.
- It is especially useful when we deal with a table that contains hierarchical data.

TYPES OF JOINS

EQUI JOIN

Specific type of a JOIN that uses the equal sign as the comparison

NON-EQUI JOIN

Specific type of a JOIN that does not use the equal sign as the comparison

NATURAL JOIN

Joins two (or more) tables based on all columns in the two tables with the same name

KEYWORD JOIN

Uses keyword for joining tables when columns have the same name in both tables

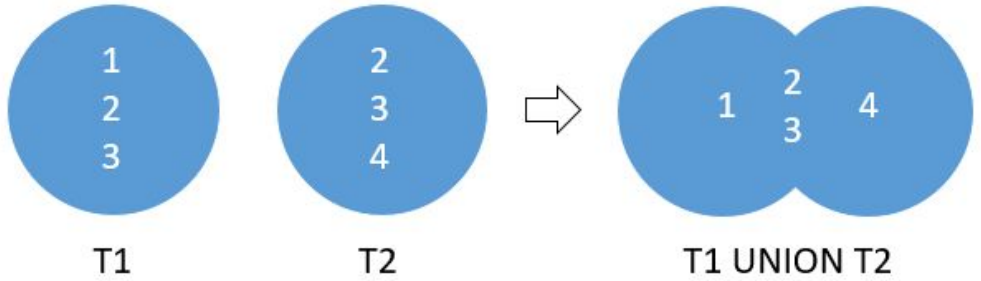
PRACTICE

- We continue practicing our JOINs queries. Let's look at the result produced by the CROSS JOIN based on our fruit tables.
- Let's create a small table with hierarchical data and perform a self join to see how useful it can be.
- Type along with your instructor ☺️.

UNION OPERATORS

- MySQL UNION operator allows to combine two or more result sets of queries into a single result set.

- By default, UNION operator removes duplicate rows even if we don't specify the DISTINCT operator explicitly.



- UNION ALL does not remove duplicate rows

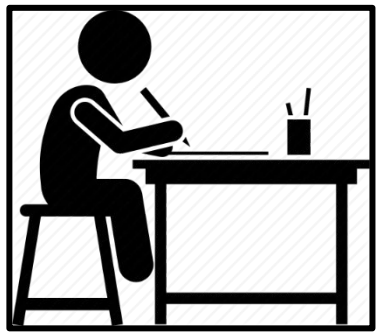
SUBQUERY

- A subquery is a nested query where the results of one query can be used in another query via a relational operator or aggregation function
- A subquery must be enclosed with parentheses
- A subquery can have only one column in the SELECT clause if used in WHERE clause
- An ORDER BY clause is not allowed in a subquery
- Subqueries can be nested within other subqueries
- Subqueries are used in WHERE, HAVING, FROM and SELECT clause

PRACTICE

- We are going to practice more by writing queries with UNION operators (based on the fruit tables).
- In addition to that, we are going to write our very first subquery using our customer database.
- Type along with your instructor ☺️.

HOMEWORK



- We will start doing this in class together as part of our practice and the task would be to complete writing the queries, i.e. finishing writing SQL code for answering these questions.

- We are going to use the database called **parts**. These queries are a bit more challenging than usual: they require joins and nested subqueries.

WRITE THE FOLLOWING QUERIES

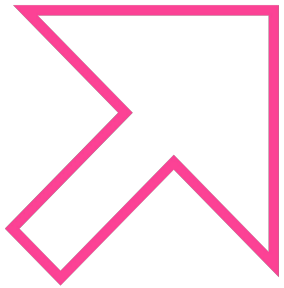
1. Find the name and status of each supplier who supplies project J2

2. Find the name and city of each project supplied by a London-based supplier

3. Find the name and city of each project not supplied by a London-based supplier

4. Find the supplier name, part name and project name for each case where a supplier supplies a project with a part, but also the supplier city, project city and part city are the same.

CODE FIRST GIRLS



REFERENCE MATERIALS

REVISION

QUICK SUMMARY

- JOINS are one of the staples in the SQL world. It is one of the most common actions to perform when working with databases.
- There are many different types of joins and they are useful in its own way (applicable for different tasks)
- JOINS make the relational model come to life by associating 2 tables together