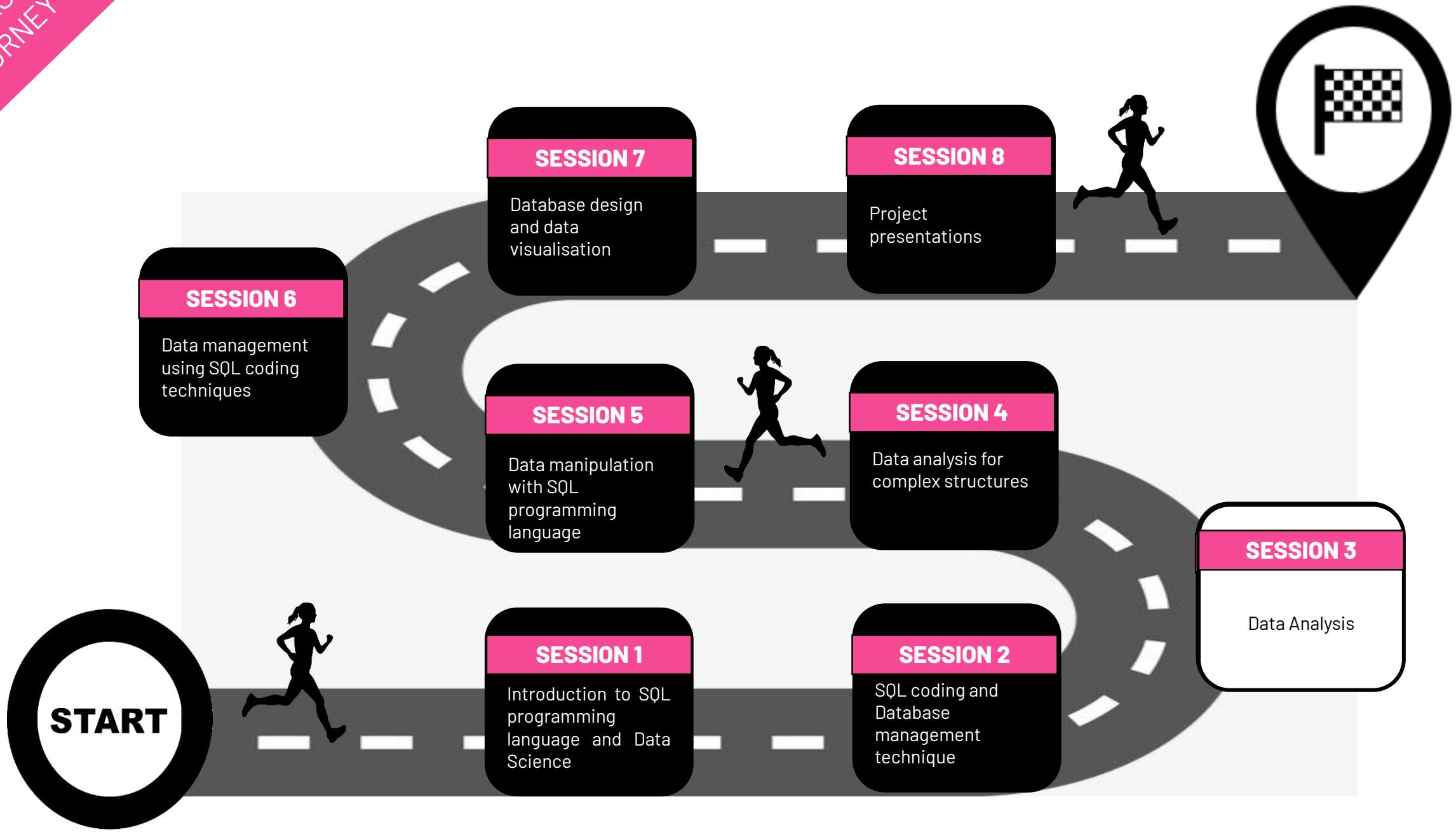
WELCOME TO CFG

YOUR INTRODUCTION TO DATABASES & SQL PROGRAMMING LANGUAGE

CODE FIRST GIRLS

TECH OPENS UP LIMITLESS OPPORTUNITIES FOR GIRLS

COURSE JOURNEY



DATA ANALYSIS

03

- **Logical operators.**
- **Data sorting, filtering and aggregation.**
- **Create more Database and Tables**
- **Learn core commands to manipulate and get data from a Database**

SYNTAX

```
SELECT
<column_name>,
<column_name>

FROM <table_name>;
```

EXAMPLE

```
SELECT
first_name,
last_name
FROM person;
```

Table Person

first_name	last_name	telephone
Lucy	Smith	020 7777 8888
Mike	Peters	020 2222 3333
Julie	Andrews	074 7878 1212

COMPARISON OPERATORS

- These are relational operators that are used to compare the value of operands (expressions) to produce a logical value. A logical value results to True or False.

- We will be using these operators in our queries to request specific set of results from a table and also to filter results.

Operator	Meaning	Example	Result
<	Less than	10<3	False
>	Greater than	10>3	True
<=	Less than or equal to	10<=3	False
>=	Greater than or equal to	10>=3	True
=	Equal to	10 = 3	False
!= OR <>	Not equal to	10!=3 or 10 <> 3	True

CONDITION KEYWORDS

- We want to ask our database more complex questions. This is why additional logical keywords are required , so that we can combine multiple expressions.

- Boolean Operators are simple logic words used as conjunctions to combine or exclude keywords in a database search.

Operator	Meaning
AND	requires both conditions to be in each result row returned.
OR	either condition (or both) will be in the returned result row.
IS NOT*	negates the condition like ‘not equals operator’, but only used for NULL.

SYNTAX

```
SELECT
<alias>.<column_name>,
<alias>.<column_name>
FROM <table_name>  <alias>;
WHERE
<alias>.<column_name> = condition
AND
<alias>.<column_name> > condition
```

EXAMPLE

Who are all the people in my class named Mary and are older than 25 years ?

```
SELECT p.name, p.surname
FROM person AS p
WHERE p.name = 'Mary'
AND p.age > 25;
```

- **SELECT clause**
- **FROM clause**
- **WHERE clause**
- **AND clause**

SYNTAX

```
SELECT
<alias>.<column_name>,
<alias>.<column_name>
FROM <table_name>  <alias>;
WHERE
<alias>.<column_name> = condition
OR
<alias>.<column_name> > condition;
```

EXAMPLE

Who are all the people in my class named Mary or their surname is Smith ?

```
SELECT p.name, p.surname
FROM person p
WHERE p.name = 'Mary'
OR p.surname = 'Smith';
```

- **SELECT clause**
- **FROM clause**
- **WHERE clause**
- **OR clause**

SYNTAX

```
SELECT
<alias>.<column_name>,
<alias>.<column_name>
FROM <table_name>  <alias>;
WHERE
<alias>.<column_name> IS NOT NULL;
```

EXAMPLE

Who are all the people in my class who have email addresses?

```
SELECT p.name, p.surname
p.email_address
FROM person p
WHERE p.email_address IS NOT NULL
```

- **SELECT clause**
- **FROM clause**
- **WHERE clause**

PRACTICE

For this practice exercise, please use our database '**bakery**'

WRITE THE FOLLOWING QUERIES

- Find all savoury items that have either pork or beef filling.
- Find all sweet items that cost 50 cents or cheaper.
- Find all sweet items and their price, except for cannoli.

LOGICAL OPERATORS

- These operators are a bit less common, but extremely useful for, retrieving, searching and analysing data.

- For example, we can use some of these operators to identify data patterns or to search for results even if we do not have the exact search criteria.

Operator	Meaning
BETWEEN (NOT BETWEEN)	selects values within a given range (excludes values within given range)
LIKE	Logical operator that determines if a character string matches a specified pattern
IN	allows you to specify multiple values in a WHERE clause
IS	Special operator, very much like ‘equals’, but is only used for values that potentially might be NULL

SYNTAX

```
SELECT
<alias>.<column_name>,
<alias>.<column_name>
FROM <table_name>  <alias>;
WHERE
<alias>.<column_name>
BETWEEN
Value1 AND value2;
```

EXAMPLE

Who are all the customers in our contact list that have placed an order up to 5 times?

```
SELECT c.name, c.surname
FROM customers c
WHERE c.orders
BETWEEN 1 AND 5;
```

- **SELECT clause**
- **FROM clause**
- **WHERE clause**
- **BETWEEN clause**

LIKE OPERATOR

- We use LIKE to search for a specific pattern in a column value.
- There are two important wildcards, which are often used in conjunction with the LIKE operator. These wildcards help us to build a pattern expression.
- % - means zero, one or however many characters
- _ - means only one single character

LIKE operator	Description
Name LIKE 'm%'	Any word/value that starts with 'm'
Name LIKE '%m'	Any word/value that ends with 'm'
Name LIKE '%or%'	Any word/value that have "or" in any position
Name LIKE '_m%'	Any word/value that have 'm' in the second position
Name LIKE 'm__%'	Any word/value that starts with 'm' and is at least 3 characters long
Name LIKE 'm%t'	Any word/value that starts with 'm' and ends with 't'

SYNTAX

```
SELECT
<alias>.<column_name>,
<alias>.<column_name>
FROM <table_name> AS <alias>;
WHERE
<alias>.<column_name>
LIKE 'M%';
```

EXAMPLE

Who are all the customers in our contact list, with first name starting with the letter 'M'?

```
SELECT c.name, c.surname
FROM customers AS c
WHERE c.name
LIKE 'M%';
```

- **SELECT clause**
- **FROM clause**
- **WHERE clause**
- **LIKE**

SYNTAX

```
SELECT
<alias>.<column_name>,
<alias>.<column_name>
FROM <table_name>  <alias>;
WHERE
<alias>.<column_name>
IN (value1, value2, value2);
```

EXAMPLE

Who are all the customers in our contact list who are named Mary, Julie, Katie or Jo?

```
SELECT c.name, c.surname
FROM customers c
WHERE c.name
IN ('Mary', 'Julie', 'Katie', 'Jo')
```

- **SELECT clause**
- **FROM clause**
- **WHERE clause**
- **IN clause**

SYNTAX

```
SELECT
<alias>.<column_name>,
<alias>.<column_name>
FROM <table_name>  <alias>;
WHERE
<alias>.<column_name> IS NULL;
```

EXAMPLE

Who are all the customers in our contacts that do not have an email address?

```
SELECT
p.surname, p.email_address
FROM person p
WHERE p.email_address IS NULL
```

- **SELECT clause**
- **FROM clause**
- **WHERE clause**
- **OR clause**

PRACTICE

For this practice exercise, please use our database '**bakery**'

WRITE THE FOLLOWING QUERIES

- Find all sweet items that start with the letter 'c'
- Find all savoury items that cost more than £1, but less than £3

AGGREGATION AND ORDER

- Sometimes we want our result data to be presented in a different way to a simple set of values returned by SELECT statement

- We don’t just want to extract data, but most importantly, we want to:
  - organize
  - understand
  - interpret
  - analyse

- We are going to learn how to shape our result set with very useful **ORDER BY** and **GROUP BY** statements.

ORDER BY

- This clause is used with SELECT statement for arranging retrieved data in sorted order.
- ORDER BY comes after WHERE clause (if we have it in a query)
- We need to specify one or more columns to order our result set by
- Two keywords that we will often use with this statement are ASC and DESC
- If we do not specify our preference explicitly, the default sorting order is always ASC.

SYNTAX

NB: if we have a column in the ORDER BY clause, it must be present in our SELECT clause

```
SELECT
<alias>.<column_name>,
<alias>.<column_name>
FROM <table_name>  <alias>
ORDER BY <alias>.<column_name> ;
```

EXAMPLE

Who are all the customers in our contacts ordered by surname?

```
SELECT
p.surname, p.name
FROM person p
ORDER BY p.surname;
```

- **SELECT clause**
- **FROM clause**
- **ORDER BY clause**

SET FUNCTIONS

- A group of ‘helper’ functions that enables us to ask more interesting, complex questions when querying a database.
- A set function is used *in place* of columns in SELECT clause. We need to pass a column name to a set function.
- These functions compute a new value from passed in column values.
- We often use DISTINCT qualifier with set functions.

SET FUNCTIONS

- Examples of Core Set Functions

Function	Effect
COUNT	Count of all column values specified (includes NULL values if * is used)
MAX	Maximum value of the column (does not include NULL values)
MIN	Minimum value of the column (does not include NULL values)
SUM	Sum of all the values of the column (does not include NULL values, only numeric column)
AVG	Average of all values of the column (does not include NULL values, only numeric column)
GROUP_CONCAT	Transforms values from a group of rows into a delimited string

- FYI: there are many more functions available, for example:

  Trigonometric and Logarithmic Functions - often used for data analysis
  sin, cos, tan, asin, acos, atan, sinh, cosh, tanh, log(<base>, <value>), ln(<value>)

SYNTAX

```
SELECT
SUM(<alias>.<column_name>)
FROM <table_name>  <alias>;
```

EXAMPLE

What is the total number of all orders placed by our customers?

```
SELECT
SUM(c.orders)
FROM customer c;
```

- **SELECT clause with the**
- **SUM SET FUNCTION**
- **FROM clause**

SYNTAX

```
SELECT
COUNT(DISTINCT
<alias>.<column_name>)
FROM <table_name>  <alias>;
```

EXAMPLE

What is the count of unique first names among our customers contacts?

```
SELECT
COUNT(DISTINCT c.name)
FROM customers c;
```

- **SELECT clause with the**
- **COUNT SET FUNCTION + DISTINCT QUALIFIER**
- **FROM clause**

GROUP BY

- It is a SQL command that is used to group rows that **have the same values**
- Allows multiple columns with a set function
- Breaks result set into subsets, runs set function against each of those subsets: result set returns 1 row per subset
- Column(s) specified in GROUP BY must appear in the SELECT LIST
- Main objective of GROUP BY is to summarise data from the database.

SYNTAX

```
SELECT
COUNT(<alias>.<column_name>),
<alias>.<column_name>

FROM <table_name> AS <alias>
GROUP BY <alias>.<column_name>;
```

EXAMPLE

What is the count of unique first names among our customers contacts?

```
SELECT
COUNT(DISTINCT c.name),
c.name
FROM customers AS c
GROUP BY c.name;
```

- SELECT clause with the
- COUNT SET FUNCTION

- FROM clause
- GROUP BY COLUMN in SELECT LIST
- GROUP BY CLAUSE

HAVING

- The HAVING clause was added to SQL because the WHERE keyword **cannot not be used with aggregate functions**.
- HAVING filters records that work on summarised GROUP BY results.
- NB: WHERE and HAVING clauses can be in the same query.
- The difference is that HAVING applies to summarised group records, whereas WHERE applies to individual records.

SYNTAX

```
SELECT
COUNT(DISTINCT<alias>.<column_name>),
<alias>.<column_name>

FROM <table_name> AS  <alias>
GROUP BY <alias>.<column_name>

HAVING COUNT(DISTINCT
<alias>.<column_name>) >= value;
```

EXAMPLE

What is the count of unique first names among our customers contacts that appear at least 3 times?

```
SELECT
COUNT(DISTINCT c.name),
c.name
FROM customers AS c
GROUP BY c.name
HAVING COUNT(c.name) >=3;
```

- **SELECT clause with the**
- **COUNT SET FUNCTION**

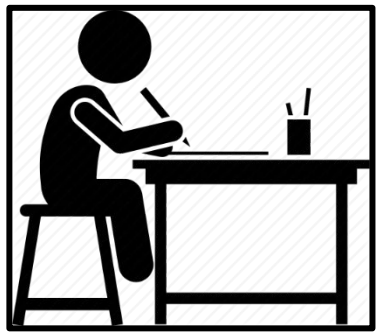
- **FROM clause**
- **GROUP BY COLUMN in SELECT LIST**
- **HAVING CLAUSE**

PRACTICE

- We are going to use a mock database that holds a table, which contains sales data for a shop chain (you created it at home).
- Importantly , we are going to assume the role of a Business Analyst or an Auditor (who do you prefer to be ☺) to work with this sales database.
- Our task will be to write a number of queries to analyse sales data and 'report it back to our big bosses'
- All queries that we will not finish in class, will need to be completed as part of homework for week 3.

HOMEWORK

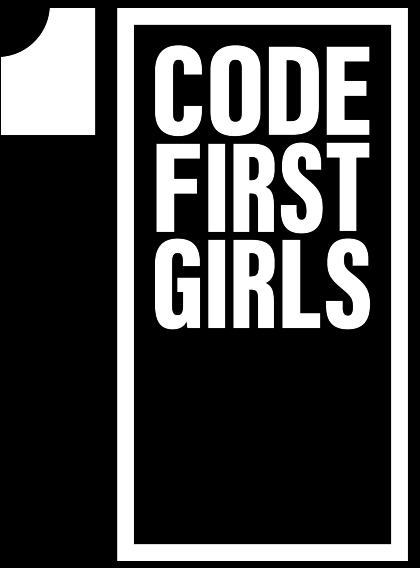
(start exercises in class – complete the rest at home)



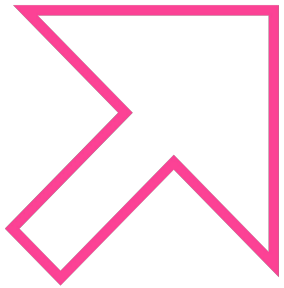
WRITE THE FOLLOWING QUERIES

- Find **ALL sales records** (and all columns) that took place in the **London store**, **not in December**, but sales concluded by **Bill or Frank** for the amount **higher than £50**
- Find out **how many sales took place each week** (in no particular order)
- Find out **how many sales took place each week AND present data by week in descending and then in ascending order**
- Find out how many sales were recorded **each week on different days of the week**
- We need to **change** salesperson's **name Inga** to **Annette**
- Find out **how many sales did Annette do**
- Find the **total sales amount by each person by day**
- How much (**sum**) **each person sold for the given period**
- How much (**sum**) **each person sold for the given period,** including the **number of sales per person**, their **average**, **lowest** and **highest sale amounts**
- Find **the total monetary sales amount achieved by each store**
- Find **the number of sales by each person if they did less than 3 sales for the past period**
- Find **the total amount of sales by month where combined total is less than £100**

THANK YOU
HAVE A GREAT WEEK!



CODE FIRST GIRLS



REFERENCE MATERIALS

REVISION

QUICK SUMMARY

- With the help of Logical Operators, Booleans and Conditional Keywords we can answer more complex questions by mean of writing more complex queries
- ORDER BY sorts result sets
- SET functions roll-up or slice result sets
- GROUP BY aggregates data and creates subsets
- HAVING restricts GROUP BY