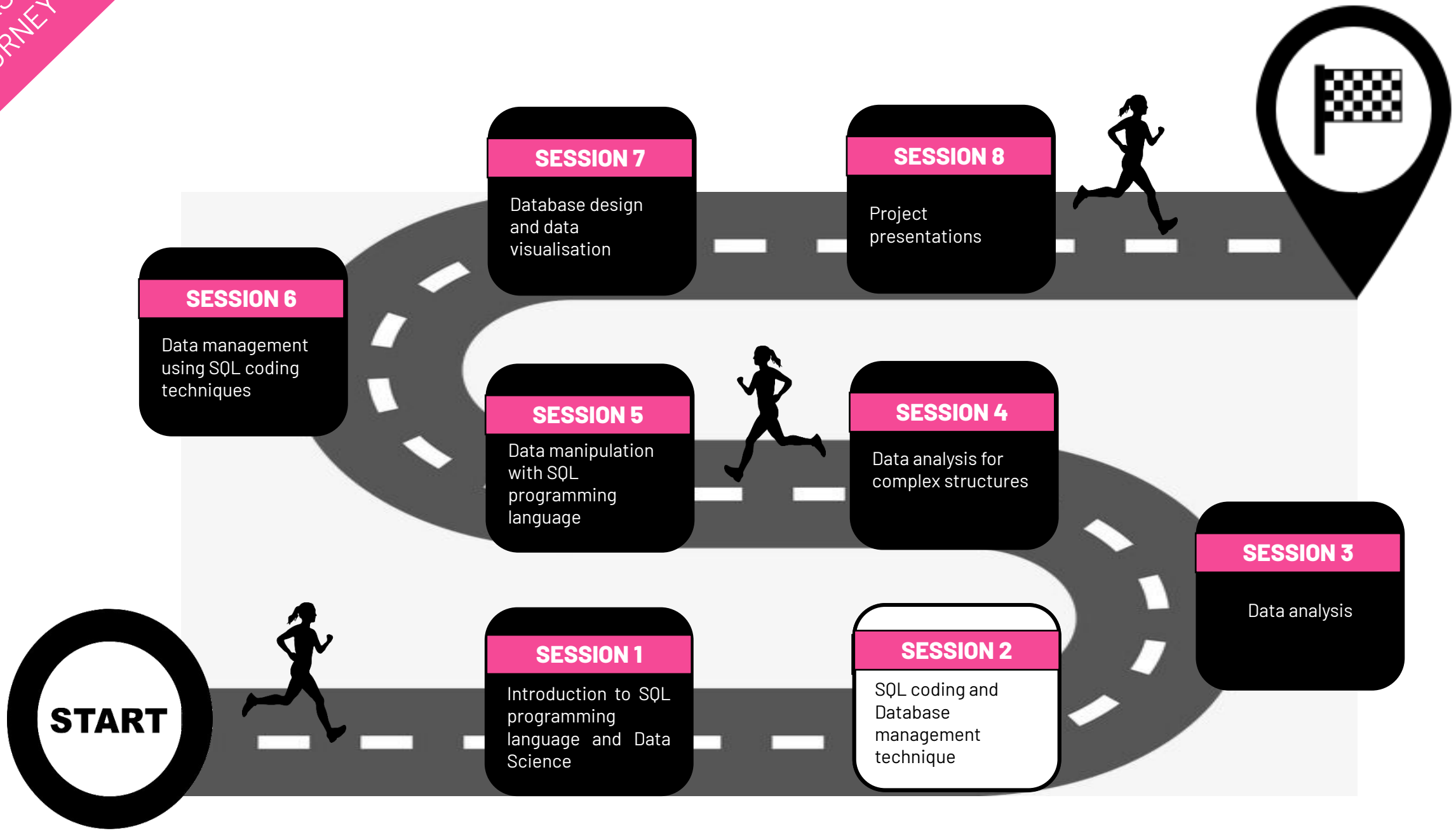
WELCOME TO CFG

YOUR INTRODUCTION TO DATABASES & SQL PROGRAMMING LANGUAGE

CODE FIRST GIRLS

TECH OPENS UP LIMITLESS OPPORTUNITIES FOR GIRLS

COURSE JOURNEY



SQL CODING
DB MANAGEMENT
TECHNIQUE

02

1. DB design and management:
   - **Normalisation**
   - **Constraints on a table**
   - **Primary and Foreign keys**
   - **Index**

2. SQL Coding:
   - **Data Modification Techniques**
   - **Data Retrieval Techniques** (SELECT statements)

PART 1: DB DESIGN AND MANAGEMENT

DATABASE NORMALISATION

- The idea behind normalisation is to organise a database into tables in such way that a table is created about one specific topic only.

- The main reasons to normalise a database are:
  - to minimise duplicated data,
  - to minimize or avoid data modification issues
  - to simplify queries

- There are three common forms of database normalization:
  - 1st, 2nd, and 3rd normal form or
  - 1NF, 2NF, and 3NF respectively

NB please read about 3 forms of normalisation:
https://www.complexsql.com/database-normalization/

NORMALISATION EXAMPLE 1

- We want to design our DB that it can answer as many 'questions' as possible including the ones we may ask in the future. We want to design it in a way that we can always add more data to it and easily modify existing data if necessary.

Table 1 – NO normalisation applied

Employee No	Employee Name	Department
1	Mary	FINANCE,TAX
2	Edith	HR
3	Anna	ADMIN

Table 2.1 – normalised

Employee No	Employee Name
1	Mary
2	Edith
3	Anna

Table 2.2 – normalised

Employee No	Department
1	FINANCE
1	TAX
2	HR
3	ADMIN

NORMALISATION EXAMPLE 2

Customer Name	Customer Address	Customer Tel No.	Product Name	Unit Cost	Quantity	Total Cost
Alex Wilson	1318 Scenic Avenue, Bothel	697-555-0142	Men's Sports Shorts, S	15.5	2	31
Alex Wilson	1318 Scenic Avenue, Bothel	697-555-0142	Water Bottle - 30 oz	1.5	3	4.5
Alex Wilson	1318 Scenic Avenue, Bothel	697-555-0142	LL Mountain Handlebars	19.76	2	39.52
Emily Brown	628 Muir Road, Los Angeles	708-555-0141	Long-Sleeve Logo Jersey, S	38.49	1	38.49
Emily Brown	628 Muir Road, Los Angeles	708-555-0141	Sport-100 Helmet, Black	13.08	2	26.16
Emily Brown	628 Muir Road, Los Angeles	708-555-0141	LL Mountain Handlebars	19.76	3	59.28

source: http://salinaitmind.blogspot.com/2012/10/data-warehouse-data-normalization.html

Normalisation

*"...make sure that every attribute is **single valued** and **provides a fact completely** and **only** about its primary key"*

- **Single Valued**
  - The attribute can only contain one piece of information
  - E.g. Polly, Moore should be two attributes – first name and surname

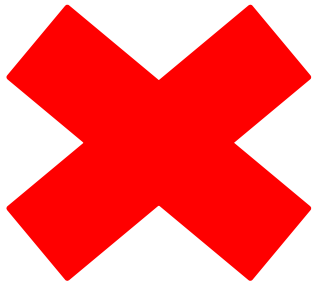
- **Provides a fact completely**
  - A given Primary Key should only return no more than one of every attribute
  - E.g. ID1 (the PK) should only return one row

- **Only about its primary key**
  - Each attribute must provide a fact about the PK and nothing else.
  - E.g. if we have a field that tells us when the room was last booked, and a flag stating when this was last booked, it becomes redundant

First Normal Form (1NF)

Each attribute contains only one value and cannot be broken down into anything smaller

Name	Address
Polly Moore	123 Corn Street, Bristol, BS1 6EB
Chloe Moore	1 Kilmarnock Street, Glasgow, GL1 6BE

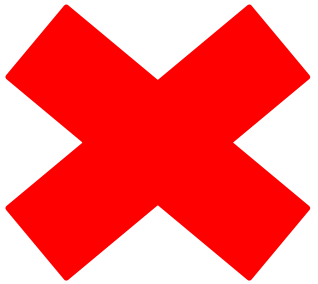


First Name	Surname	Number	Street	City	Postcode
Polly	Moore	123	Corn St	Bristol	BS1 6EB
Chloe	Moore	1	Kilmarnock St	Glasgow	GL1 6BE

Second Normal Form (2NF)

Cannot contain any partial Dependencies

ID	First Name	Surname	Course	Course Cost
1	Polly	Moore	Data Science	£100
2	Chloe	Moore	Data Engineering	£240
3	Polly	Moore	Data Engineering	£240



CourseID	Course	Cost
1	Data Science	£100
2	Data Engineering	£200

ID	Forename	Surname	Course
1	Polly	Moore	2
2	Chloe	Moore	2
3	Polly	Moore	1

Third Normal Form (3NF)

All Non-Primary Fields are dependent on the Primary Key

ID	Forename	Surname	Subject	Location	Country
1	Polly	Moore	Data Engineering	Bristol	England
2	Chloe	Moore	Data Science	Glasgow	Scotland
3	Rachael	Moore	Data Analysis	Inverness	Scotland

Country is dependent on location, not the ID

Third Normal Form (3NF)

All Non-Primary Fields are dependent on the Primary Key

ID	Forename	Surname	Subject	Location	Country
1	Polly	Moore	Data Engineering	Bristol	England
2	Chloe	Moore	Data Science	Glasgow	Scotland
3	Rachael	Moore	Data Analysis	Inverness	Scotland

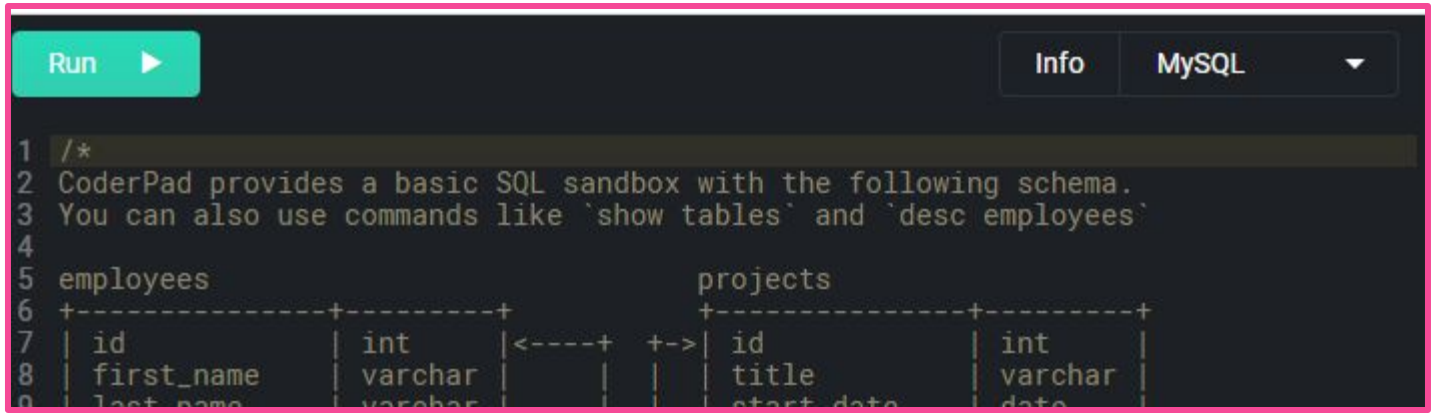
ID	Forename	Surname	Subject
1	Polly	Moore	Data Engineering
2	Chloe	Moore	Data Science
3	Rachael	Moore	Data Analysis

Location	Country
Bristol	England
Glasgow	Scotland
Inverness	Scotland

PRACTICE

LET'S REVIEW THE DB DESIGN IN OUR SANDBOX

https://coderpad.io/sandbox



SQL CONSTRAINT TYPES

- Constraints are the rules that we can apply on the type of data in a table.
- In other words, we can specify the limit on the type of data that can be stored in a particular column in a table. Using constraints ensures the accuracy and reliability of the data in the table.
- If there is a mismatch or any violation between the constraint we set and the data, then the action that we are trying to perform would be aborted.

NB please read about constraints: https://www.studytonight.com/dbms/sql-constraints.php

PRIMARY KEY

- primary key is a single field or combination of fields that **uniquely defines a record**

ONLY ONE PRIMARY KEY IN A TABLE

- Must be NOT NULL
- Can be a multiple columns (compound key)
- Can be defined in either a CREATE TABLE statement or an ALTER TABLE statement

NULL	NOT NULL
Default for a column definition	Must be specified on column definitions
It means that we are allowed to insert NULL values	It means we are not allowed to insert NULL values. Inserting a NULL would raise an error!

SYNTAX

```
CREATE TABLE <table_name>
(col1 Type KEY DEFINITION,
 col2 Type,
 col3 Type);
```

EXAMPLE

```
CREATE TABLE customers
(customer_id INTEGER PRIMARY KEY,
 name VARCHAR(50),
 surname VARCHAR(50) NOT NULL,
 telephone INTEGER);
```

- **CREATE TABLE + TABLE NAME**
- **PRIMARY KEY**
- **COLUMNS**

SYNTAX

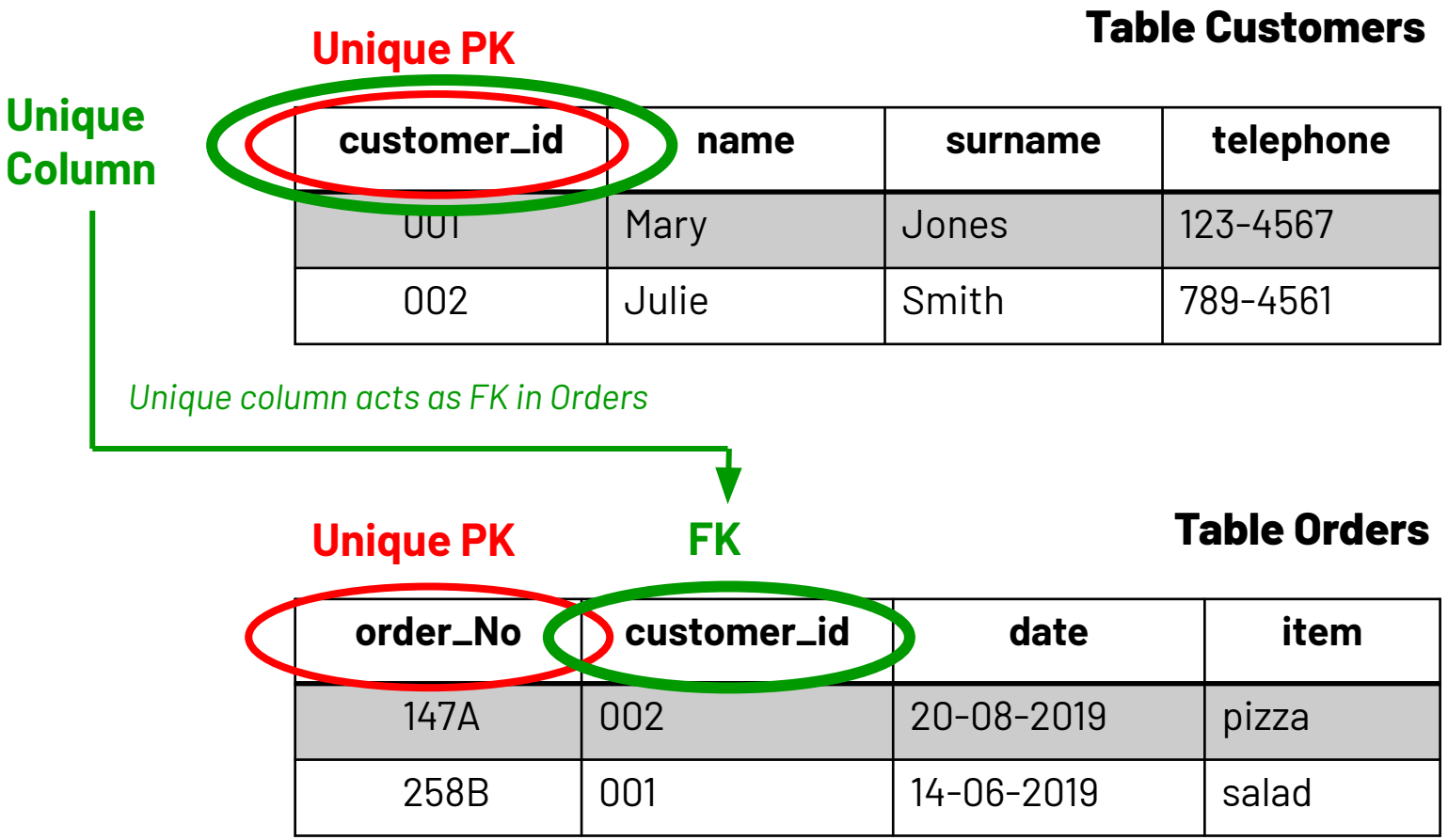
```
CREATE TABLE <table_name>
(col1 Type,
 col2 Type,
 col3 Type,
 CONTRAINT
 <constraint_name>
 <constraint_type>
(<col_that_it_applies_to>)
);
```

EXAMPLE

```
CREATE TABLE customers
(customer_id INTEGER,
 name VARCHAR(50),
 surname VARCHAR(50) NOT NULL,
 telephone INTEGER,
CONSTRAINT
pk_ customer _id
PRIMARY KEY
(customer _id)
);
```

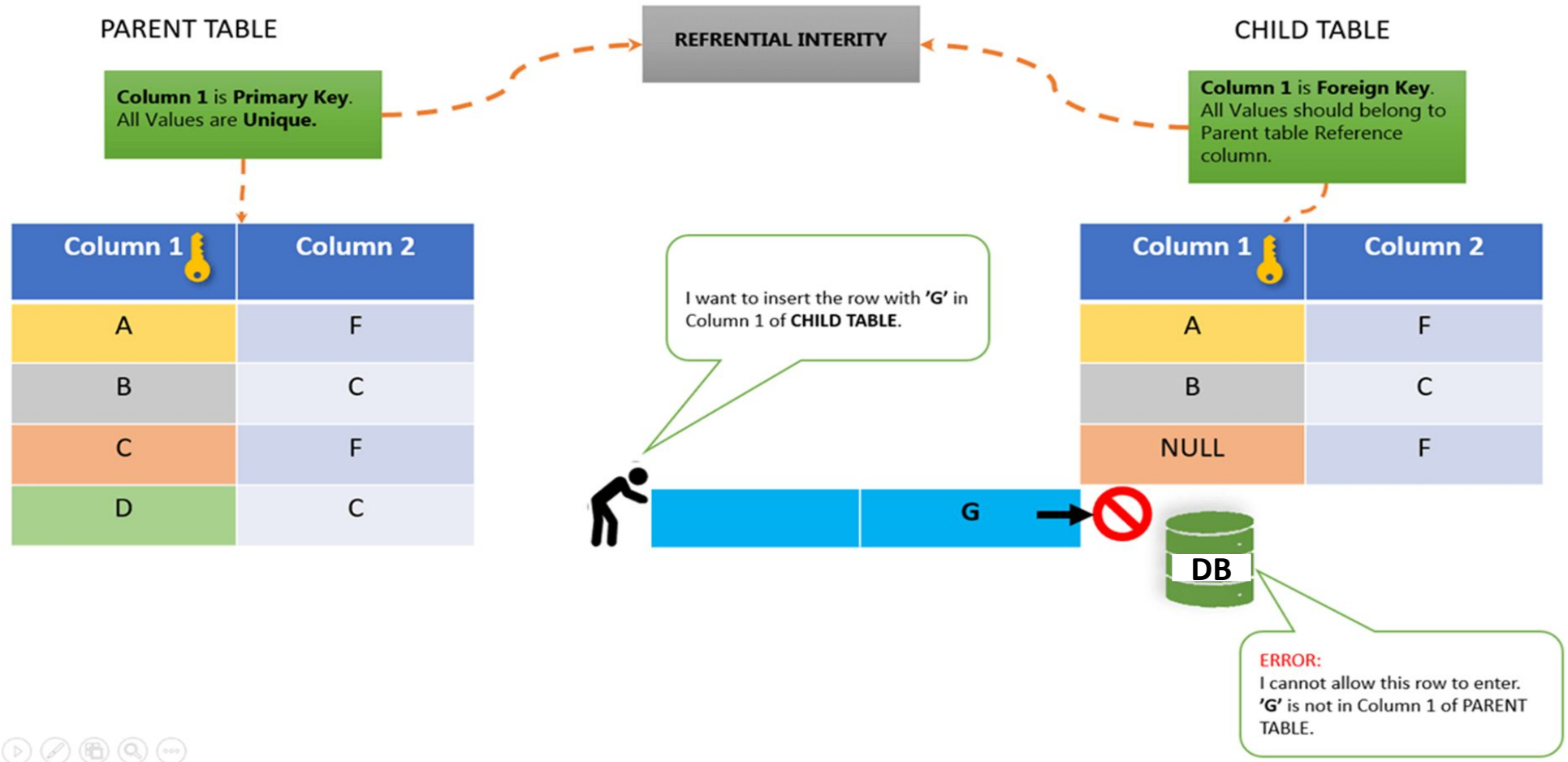
- **CREATE TABLE + TABLE NAME**
- **CONSTRAINT**

FOREIGN KEY



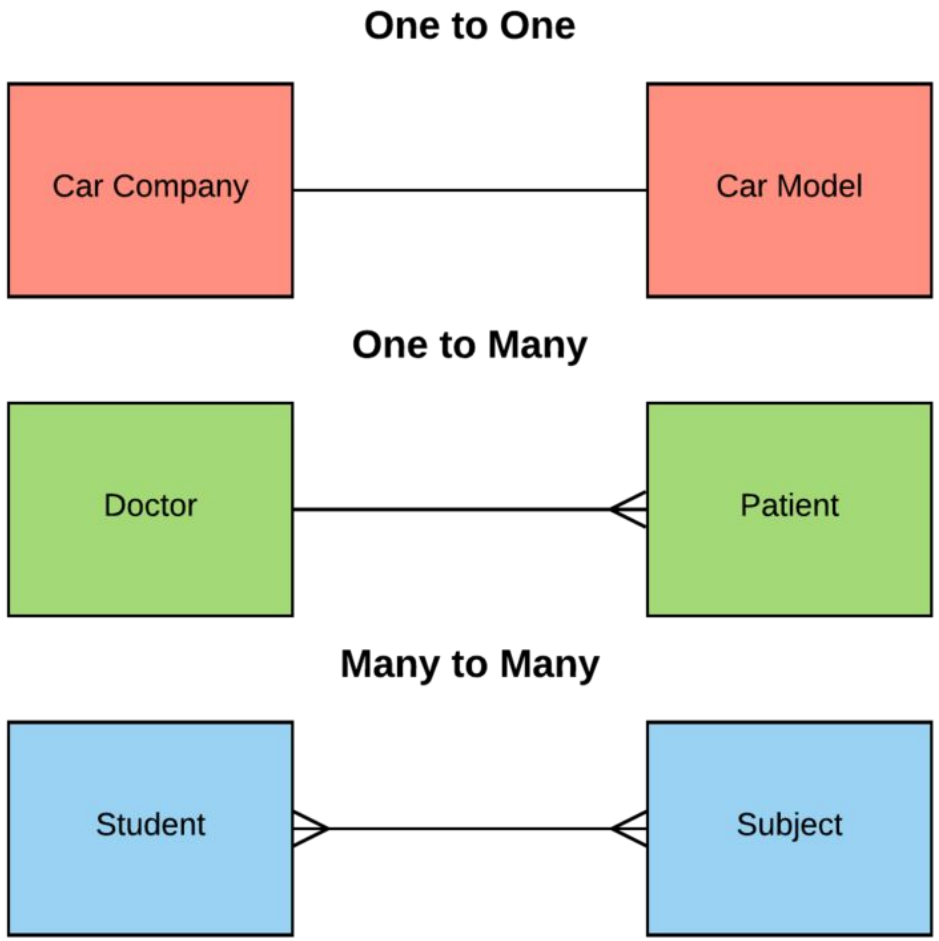
NB please read about Foreign Key: http://www.mysqltutorial.org/mysql-foreign-key/

DATA INTEGRITY



source: guru99.com

DATABASE RELATIONSHIPS



QUICK NOTE:
INDEX

- Primary keys are an example of indexes
- They are automatically set as indexes
- They are best to use with big databases instead of using a non-key or non-indexed values.
- Instead of scanning the whole table, index allow MySQL to quickly find rows with a specific value which makes the search quicker
- More about this in Reference Material

NB please read about MySQL Index:https://www.mysqltutorial.org/mysql-index/

PRACTICE

Today we are going to be pizza makers, bakers and small restaurant owners! We accept orders online or by telephone and deliver pizza to our customers.

We need to create a database to hold information about our customers, so we can keep records of their names, addresses, phone numbers, email addresses and any other useful information like placed orders.

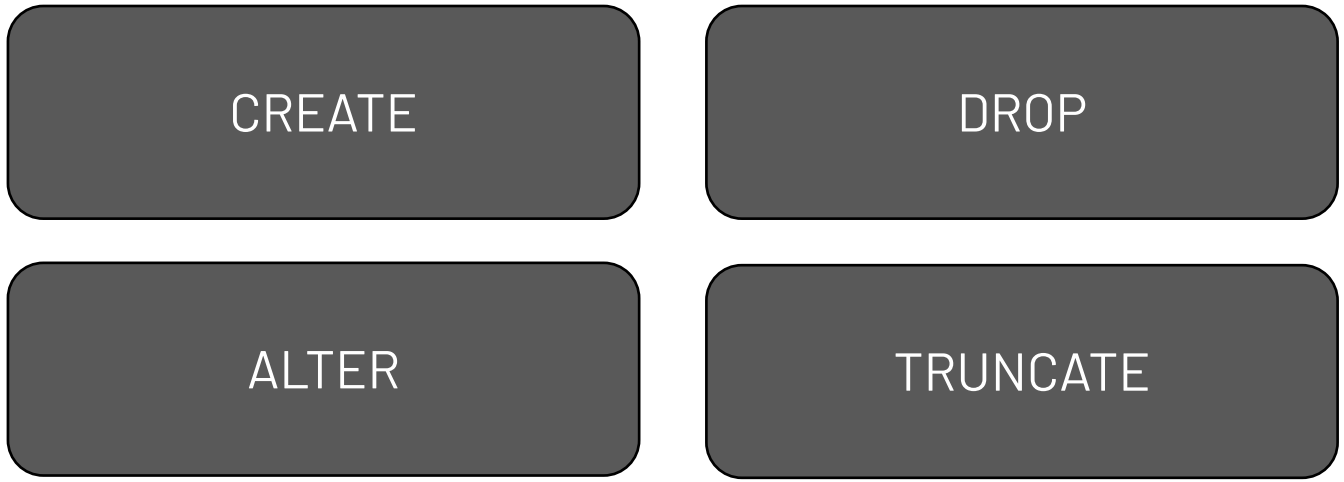
TASKS

- Design and create a relational normalised database called **customers**.
- Set reasonable **primary keys** to the tables.
- Set NOT NULL **constraints** on the columns that you think must have values.
- Let’s do it together!

PART 2: SQL CODING

CORE COMMANDS DDL

- **DDL** stands for "Data Definition Language".
- It is a subset of **SQL statements** that create the database schema and change its structure.
- Defines the columns of the table
- Typically and structural changes of the database schema refer to creating, deleting, or modifying schema objects such as databases or tables.



NB please read more about DDL: https://www.w3schools.in/mysql/ddl-dml-dcl/

CREATE TABLE

- Create a table in your database
- Specify table name
- Add column names and data types

SYNTAX

```
CREATE TABLE <table_name> (
    <col_name> <col_data type>,
    <col_name> <col_data type>,
      ...
);
```

EXAMPLE

```
CREATE TABLE customers (
      CustomerID int,
      FirstName varchar(255)
      LastName varchar(255)
);
```

DROP TABLE

- Removes table and all data from database
- BE CAREFUL!
- Error if table is a foreign key to another table

SYNTAX

DROP TABLE <table_name>;

EXAMPLE

DROP TABLE customers;

ALTER TABLE

- Used to change an existing table
- Add/remove column
- Change column data type
- Change column constraints
- Must comport with current data

SYNTAX

```
ALTER TABLE <table_name>;
RENAME TABLE
<table_new_name>;
```

EXAMPLE

```
ALTER TABLE customers;
RENAME TABLE all_customers;
```

SYNTAX

```
ALTER TABLE <table_name>
ADD CONSTRAINT
 <constraint_name>
 <constraint_type>
(<col_that_it_applies_to>)
REFERENCES
<table_name2>
(<col2_that_it_applies_to>)
);
```

EXAMPLE

```
ALTER TABLE orders
ADD CONSTRAINT
fk_customer_id
FOREIGN KEY
(customer_id)
REFERENCES
customers
(customer_id);
```

- **CREATE TABLE + TABLE NAME**
- **CONSTRAINT**

TRUNCATE TABLE

- DROP deletes the table itself.
- TRUNCATE deletes the data inside of table but not the table itself.

SYNTAX

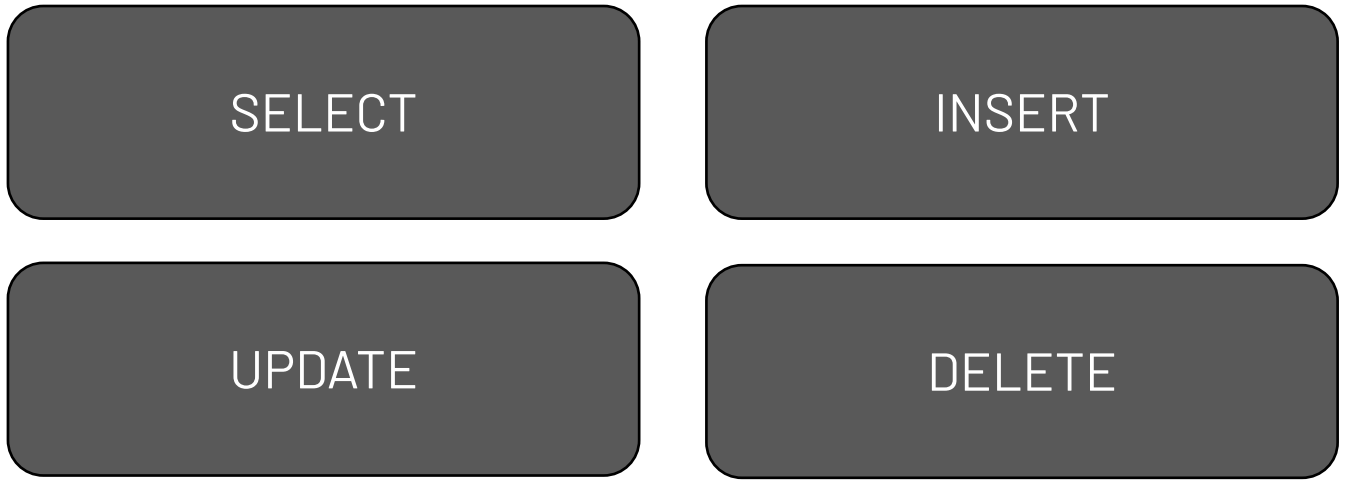
TRUNCATE TABLE <table_name>;

EXAMPLE

TRUNCATE TABLE customers;

CORE COMMANDS DML

- **DML** stands for "Data Manipulation Language".
- It is a subset of **SQL statements** that change the structure of the database schema.
- Typically adds or updates the rows ("tuple") of the table



NB please read more about DML: https://www.w3schools.in/mysql/ddl-dml-dcl/

SELECT

- SELECT which data to display and manipulate
- Use * to show all data
- FROM to specify which table to get data from

SYNTAX

```
SELECT <col_name>
FROM <table_name>;
```

EXAMPLE

```
SELECT FirstName
FROM customers;
```

- SELECT COMMAND
- TABLE NAME

```
SELECT *
FROM customers;
```

- * to SELECT all data

SELECT DISTINCT

- There are many ways that enable us to constrain the number of results returned by our query
- Using a DISTINCT Qualifier
- It is the keyword, which means that in our query we are asking for a unique set of results
- In other words we want non-repeating values in the result columns to be returned

SYNTAX

Table name Alias

SELECT DISTINCT
<alias>.<column_name>,

FROM <table_name>
AS <alias>;

EXAMPLE

What are the fist names of people in my class?

first_name
Julie
Mary
Mary
Joanna
Julie

first_name
Julie
Mary
Joanna

SELECT
p.first_name,
FROM person
AS p;

SELECT **DISTINCT**
p.first_name,
FROM person AS
p;

WHERE

- The WHERE clause is a constraint that can be applied to the result set
- The WHERE clause describes the **conditions** to match for rows to qualify for result set
- It comes after the FROM statement
- It contains **Boolean** expressions
- Only rows that match a condition are selected for the result set

SYNTAX

SELECT
<alias>.<column_name>,

FROM <table_name> AS
<alias>;

EXAMPLE

What is the surname of all my classmates who are called Mary?

```
SELECT
p.surname
FROM person AS p
WHERE p.name = 'Mary'
```

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

PRACTICE

Use the database PARTS that we created and populated at home

WRITE THE FOLLOWING QUERIES

- Using the table ‘parts’, return all unique part names. What happens if we want to return all unique parts and their id number? Why?

- Refer to the table ‘projects’ and return all projects that are run in London.

INSERT

- INSERT new data in a table

- Two ways to use INSERT:
  - Specify column names and values to insert in them
  - If values are being added to all columns, then no need to specify the column names BUT make sure the order of the values is equivalent the order of the rows.

SYNTAX

```
INSERT INTO <table_name>
(col1, col2, col3, …)
VALUES (val1, val2, val3, …)
```

```
INSERT INTO <table_name>
VALUES (val1, val2, val3, …)
```

EXAMPLE

```
INSERT INTO customers
(FirstName,LastName)
VALUES (“John”, “Doe”)
```

```
INSERT INTO customers
VALUES (654258, “Jessica”,
“Day”)
```

- **Our customer table has 3 columns: CustomerID, FirstName and LastName.**
- **CustomerID = 654258**
- **FirstName = “Jessica”**
- **LastName = “Day”**

UPDATE

- Modifies column(s) in a single table
- WHERE clause dictates which rows
- SET keyword follows table name

SYNTAX

```
UPDATE table_name
SET
table_name.col1 = new_value
WHERE
table_name.col2 = value;
```

EXAMPLE

```
UPDATE contacts
SET
contacts.mobile = 123456789
WHERE
contacts.surname = ‘Andrews’
```

- UPDATE COMMAND
- TABLE NAME
- SET KEYWORD
- VALUES
- WHERE CLAUSE

DELETE

- DELETES one or more rows in a table
- Permanent!
- DELETE FROM is actual full command
- WHERE clause is critical!

SYNTAX

EXAMPLE

DELETE FROM table_name;

- DELETE FROM customers;

- DELETE COMMAND

BAD PRACTICE ☹️

DELETE FROM table_name
WHERE
table_name.col = value;

- DELETE FROM customers
- WHERE
- customers.id = 007;

- DELETE COMMAND
- WHERE CLAUSE

GOOD PRACTICE ☺️

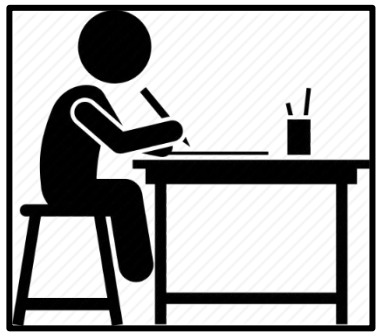
PRACTICE

We have our pizzeria customers database. Let's modify some tables in the database, so we add Foreign Keys to tables and define relationships between our tables.

TASKS

- Add some data to the tables in the **customers database**
- Alter tables **email_address** and **phone_number** in the **customers database** by adding **Foreign keys** that reference **Primary keys** from relevant tables.
- Remove the table called **orders** from our database.

HOMEWORK

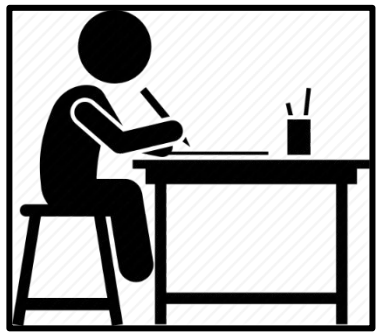


- Revise the slides to re-cap all materials from Session 2.
- Read about **Normalisation**: https://www.complexsql.com/database-normalization/
- Read about **SQL Commands**: https://www.w3schools.in/mysql/ddl-dml-dcl/
- Read about **Foreign Key**: http://www.mysqltutorial.org/mysql-foreign-key/
- Read about Constraints: https://www.studytonight.com/dbms/sql-constraints.php

USE PARTS DB TO WRITE THE FOLLOWING QUERIES

- Find the **name** and **weight** of each **red part**
- Find **all UNIQUE supplier(s) name from London**

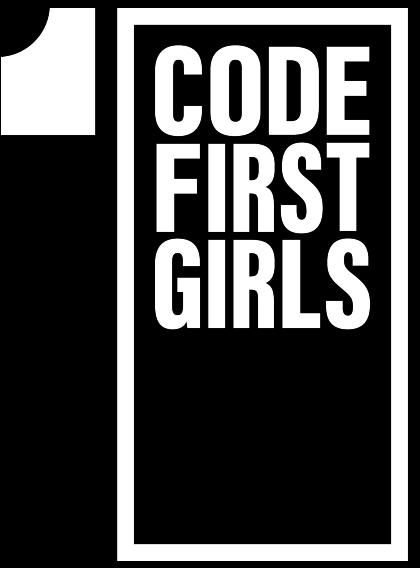
HOMEWORK



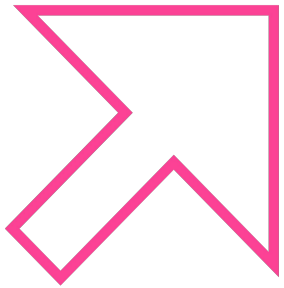
- Create a new database called **SHOP –** we will be using it during next lesson.
- Add a new table called **SALES1**. It should look like this:

Store	Week	Day	SalesPerson	SalesAmount	Month
London	2	Monday	Frank	56.25	May
London	5	Tuesday	Frank	74.32	Sep
London	5	Monday	Bill	98.42	Sep
London	5	Saturday	Bill	73.90	Dec
London	1	Tuesday	Josie	44.27	Sep
Dusseldorf	4	Monday	Manfred	77.00	Jul
Dusseldorf	3	Tuesday	Inga	9.99	Jun
Dusseldorf	4	Wednesday	Manfred	86.81	Jul
London	6	Friday	Josie	74.02	Oct
Dusseldorf	1	Saturday	Manfred	43.11	Apr

THANK YOU
HAVE A GREAT WEEK!



CODE FIRST GIRLS



REFERENCE MATERIALS

REVISION

FOREIGN KEY

- A foreign key is a field in a table that matches another field of another table. A foreign key places constraints on data in the related tables.
- A foreign key can be a column or a set of columns. The columns in the child table often refer to the primary key columns in the parent table.
- A table may have more than one foreign key, and each foreign key in the child table may refer to a different parent table.

NB please read about Foreign Key: http://www.mysqltutorial.org/mysql-foreign-key/

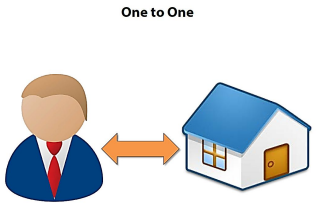
DATABASE RELATIONSHIPS

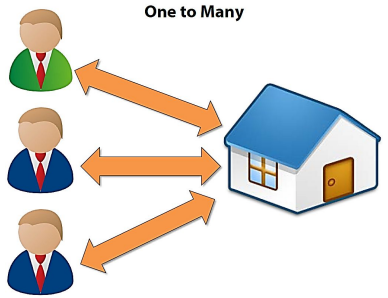
When creating a database, common sense dictates that we use separate tables for different types of entities.

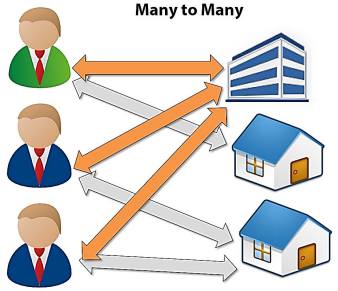
Some examples are: customers, orders, items and so on. But we also need to have relationships between these tables. For instance, customers make orders, and orders contain items.

There are several types of database relationships:

- One to One Relationships
- One to Many
- Many to One Relationships
- Many to Many Relationships







INDEX

- Think of it like a book index! Instead of reading through the whole book to find a certain word, you can look at the index and find the exact page where the word can be found
- They are best to use with big databases
- Instead of scanning the whole table, index allow MySQL to quickly find rows with a specific value which makes the search quicker
- More about Indexes: https://www.mysqltutorial.org/mysql-index/

SYNTAX

```
CREATE TABLE <table_name> (
col1 INT PRIMARY KEY,
col2 INT,
col3 INT,
INDEX (c2,c3) );
```

EXAMPLE

```
CREATE TABLE customers
(customer_id INT PRIMARY KEY,
Personal_id INT,
 name VARCHAR(50),
 surname VARCHAR(50) NOT NULL,
 telephone INT,
INDEX (personal_id) );
```

SYNTAX

```
DROP INDEX <index_name> ON
<table_name>;

DROP INDEX `PRIMARY` ON
<table_name>;
```

EXAMPLE

```
DROP INDEX person_id ON
customers;

DROP INDEX `PRIMARY` ON
customers;
```

- HOW TO DROP INDEX
- HOW TO DROP A PRIMARY KEY

The data remains but it is no longer an index/PK

QUICK SUMMARY

PRIMARY KEY

- Unique identifier of row
- One per table
- Does not allow NULL
- Single or multiple columns (composite columns)

FOREIGN KEY

- Columns in a table that refer to a Primary Key of another table
- Enforces referential integrity
- Foreign key reinforces relationships between tables:
  - One-to-one
  - One-to-many
  - Many-to-many