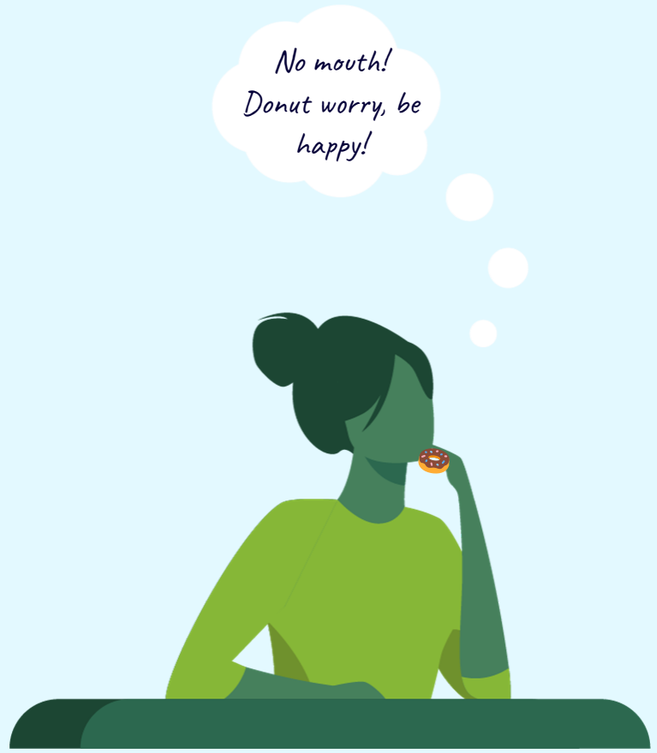
LLOYDS BANKING GROUP

Databases and MySQL

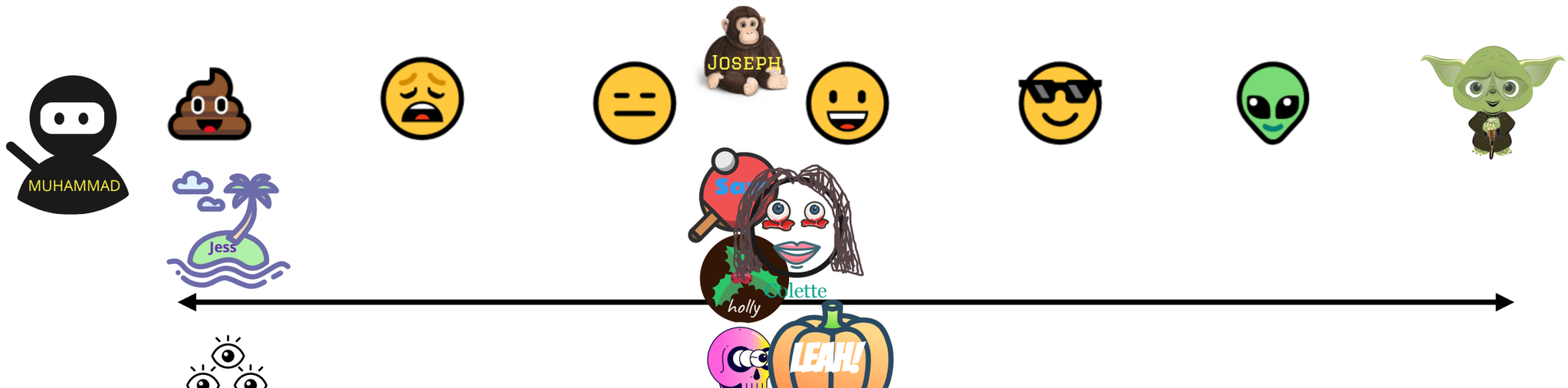
A SQL query enters a bar, walks up to two tables..
Can I join you?

Agenda

- What is a database
- Electronic database
- Relational database
- Normalisation & ERD
- RDBMS and MySQL
- SQL - DML and DDL
- Creating Tables
- Inserting Rows
- Basic SELECT query
- Selecting and limiting rows
- Aggregating rows
- Joining Tables

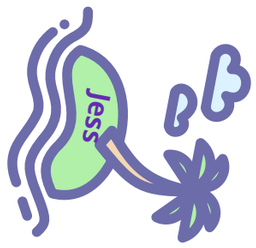


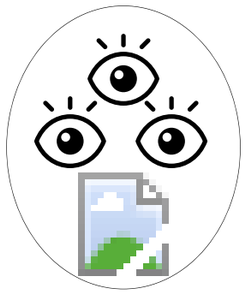
Copy & Paste your AVATAR and place it on the scale where you think your SQL knowledge is?



COPY or Create a NEW Avatar

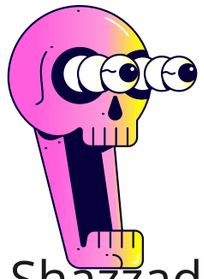
Brave Knights





Valle

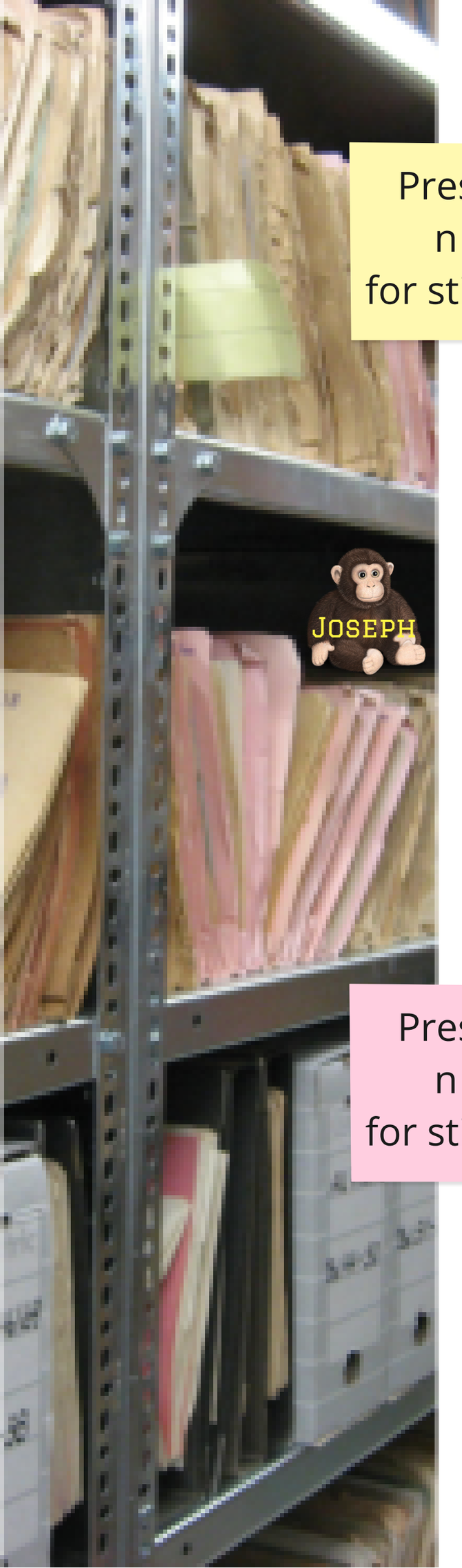
Peasants



Shazzad

Seonaid

Swamp Castle



What is your definition of a database?

Press
n
for sticky

group of structured data stored usually in tables

Kaitlin

holly

a collection of data that can be used for analysis

Place to store data

Valle

Where data is stored

Jess

a large set of numbers/text organised into tables

LEAH!

a set of data stored somewhere

Colette

Shazzad

A place where data is stored

Collection of Data

Sam

(noun) a structured set of data held in a computer, especially one that is accessible in various ways

n

Structured collection of data

MUHAMMAD

Give some examples?

Press
n
for sticky

MS SQL Server

Shazzad

holly

excel

list of items availble at Amazon

filing cabinet

Spreadsheet = SET of Data

First	Last	Job-id	Email	Salary	Dept	Mgr	Address	City
Robert	Burns		rb@gmail	300k	Exec		Argyle St	Glasgow
Billy	Connolly	Sys_Admir	billy@gmail	55k	IT	RB	Argyle St	Glasgow
Rob	Roy	IT_prog	rob@gmail	100k	IT	RB	Argyle St	Glasgow
William	Wallace	Sales_exec	willy@gmail		Sales	RB	High Street	Stirling
Kenny	Dalglish	DB_dev	kenny@gmail	250k	Finance	RR	Royal Mile	Edinburgh
Rod	Stewart	Sales_exec	rod@gmail	75k	Sales	WW	High Stree	Stirling
Julia	Donaldson	Sales_exec	julia@gmail	50k	Sales	WW	High Street	Stirling

much more data

analyse it easily

Safer & able to easily sha

easier t access. C store mo data

Press n for sticky

Pros of a worksheet

Easy

can make calculations easier

Can sort data easily

can compare between employees easily

All data in one place

saves time

Cons of a worksheet

dependent on skills of user

Limit to the amount of rows that can be stored

have to ensure its security

Easy to share with others

data can be breached

Easier to access

Less difficult to lose

Higher security

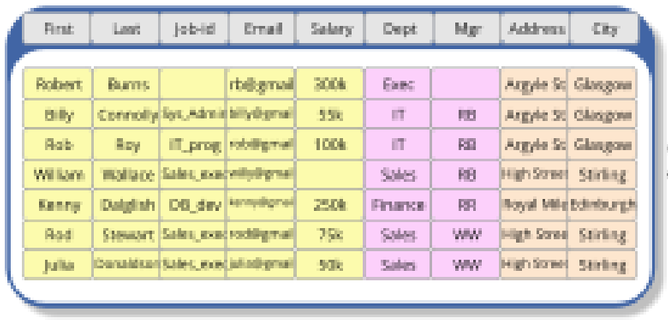
cant really store qualitative data very well

Large sets difficult to manage

errors can occur

can contain redundant data

A Relational Model of Data for Large Shared Data Banks

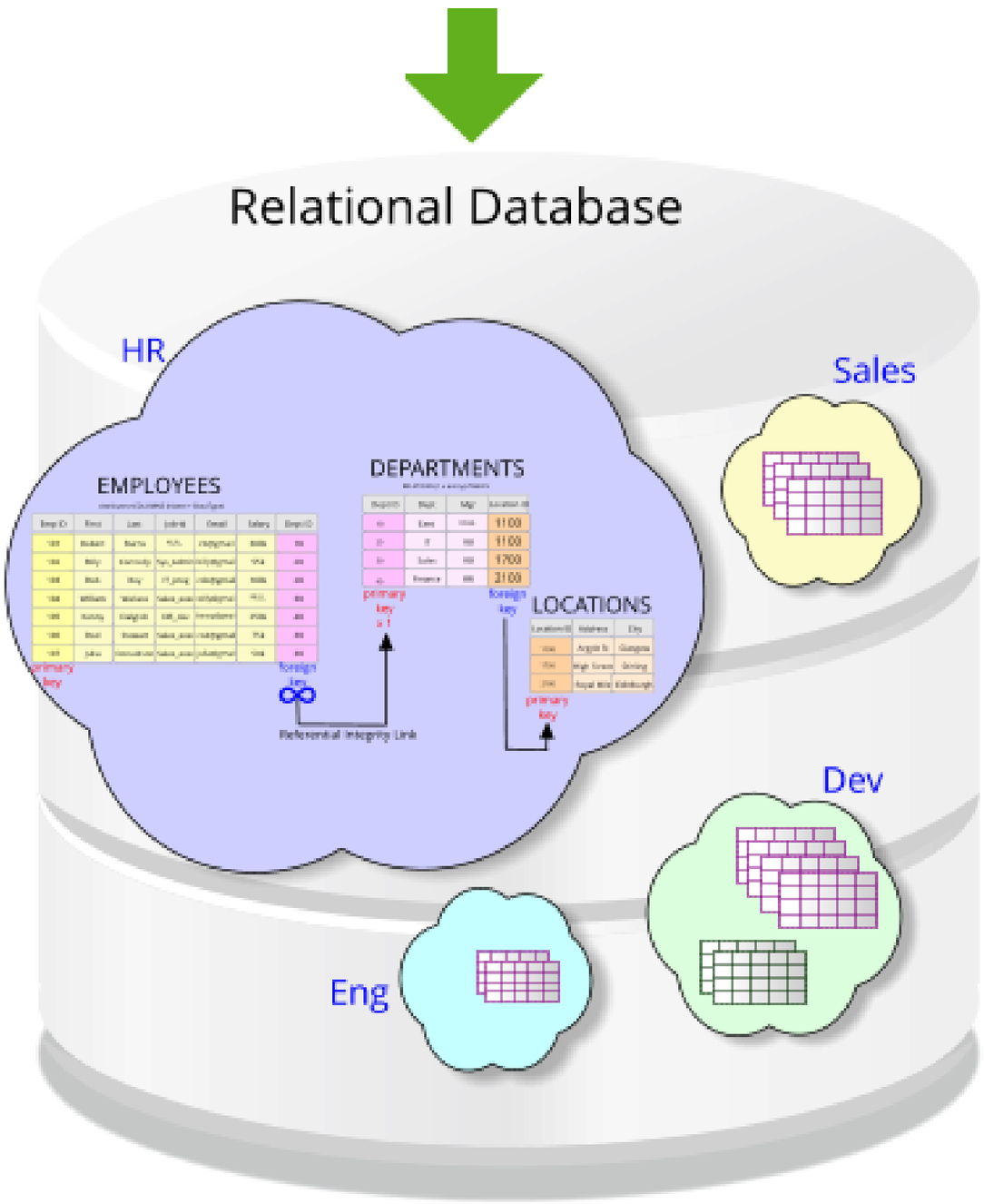


SET of Data

1971 - Dr Ted Codd (IBM) - n x ENTITIES e.g. Detail about Employees, Depts and Locations

Normalisation

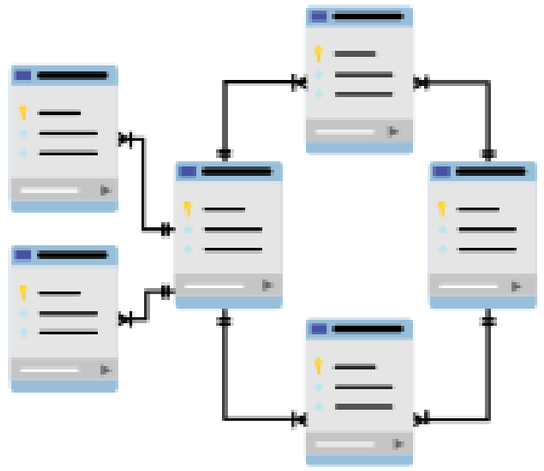
(smaller sets of data = Detail 1 x Entity and Unique KEYS



Entity Relationship Modelling

Entity Relationship

E.R Diagram



SCHEMA = Collection of Database Objects Owned by someone/something

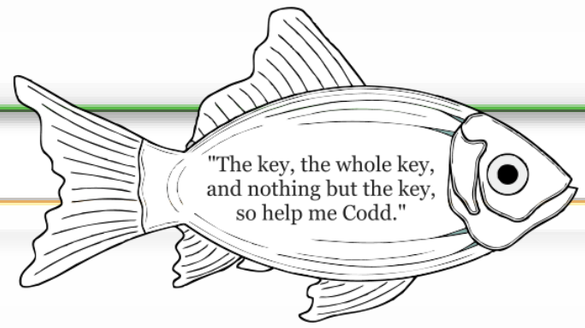
Normalisation

1NF

- Get rid of any columns that hold the same data
- Split up data that can be
- Each Row must be unique

2NF

- Get rid of data not dependant on EVERY part Primary Key



3NF

- Keep Splitting it up.
- No Non-key attribute should be dependant on another non-key attribute

- Divide 'Large' set into smaller sets (**ENTITY**)
- 1 x Entity per set (rectangle) = table
- **ATTRIBUTES** = Columns (name + datatype)
- Enforce uniqueness onto rows using a Primary Key
- Entities can have **RELATIONSHIPS** with entities using Foreign Keys
    - 1-to-1 = 1 husband to 1 wife
    - many-to-1 = many students to 1 school
    - 1-to-many = 1 customer to many bank accounts
    - many-to-many = many students to many teachers
- Columns can be optional (NULLs allowed)

EMPLOYEES

Attributes=COLUMNS (Name + DataType)

Emp ID	First	Last	Job-id	Email	Salary	Dept ID
101	Robert	Burns	NULL	rb@gmail	300k	10
102	Billy	Connolly	Sys_Admin	billy@gmail	55k	20
103	Rob	Roy	IT_prog	rob@gmail	100k	20
104	William	Wallace	Sales_exec	willy@gmail	NULL	30
105	Kenny	Dalglish	DB_dev	kenny@gmail	250k	40
106	Rod	Stewart	Sales_exec	rod@gmail	75k	30
107	Julia	Donaldson	Sales_exec	julia@gmail	50k	30

primary key

foreign key

DEPARTMENTS

RELATIONS (1 x entity)=TABLES

Dept ID	Dept	Mgr	Location ID
10	Exec	NULL	1100
20	IT	RB	1100
30	Sales	RB	1700
40	Finance	RR	2100

primary key x 1

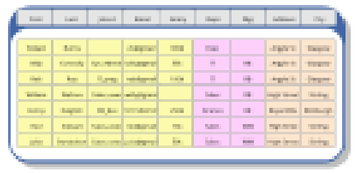
foreign key

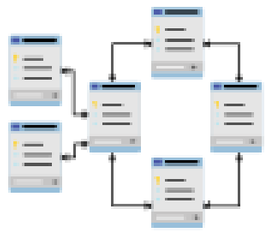
LOCATIONS

Location ID	Address	City
1100	Argyle St	Glasgow
1700	High Street	Stirling
2100	Royal Mile	Edinburgh

primary key

Referential Integrity Link



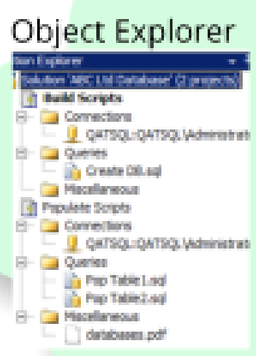


Entity Relationship Modelling
Entity Relationship
E.R Diagram

Program

GUI Tool

Object Explorer



Begin
SQL
end;

CLI Tool

RDBMS

SQL Engine

Relational Database

HR

EMPLOYEES

DEPARTMENTS

LOCATIONS

primary key

foreign key

Referential Integrity Link

Sales

Dev

Eng

Microsoft GNU Linux

x86-64/AMD/Arm/Sparc/PPC

Structured Query Language

Data management langua

data manipulation language

database modification language

data something

data definition language

data development language

definition

data control language

Dominic Calvert-Lewin

transaction control language

DML insert, update, delete, merge, select

DDL create, alter, drop, truncate

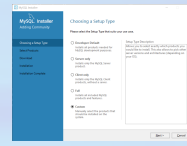
DCL grant, revoke

TCL commit, rollback, savepoint

Installing MySQL

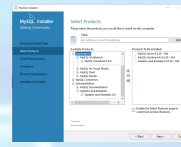
Download the installer - mysql-installer-community
- Run the installer
- Accept prompts

Setup Type - choose CUSTOM and NEXT



Select Products - user green arrows to select:
- Newest MySQL Server from MySQL Servers
- Newest MySQL Workbench from Applications
- Samples and Example from Documentation

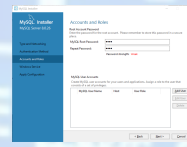
Click NEXT and EXECUTE to download
Once installed click NEXT



Product Configuration - click NEXT
Type and Networking - click NEXT
Authentication Method - click NEXT

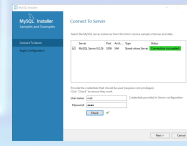
Account and Roles
- Enter password for root (admin) account
    - Min 4 chars - ENTER lotr

Click NEXT



Windows Service - click NEXT
Server File Permissions - click NEXT
Apply Configuration - click EXECUTE
Click FINISH
Product Configuration - click NEXT

Connect to Server
- ENTER PASSWORD - click CHECK
- If successful - click NEXT



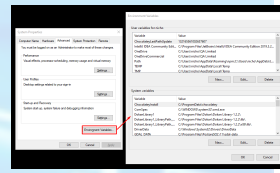
Apply Configuration - click Execute & FINISH
Product Configuration - click NEXT
Installation complete - click FINISH

MySQL Workbench starts

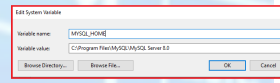
Configuring Env Variables

In Windows START menu, search for ENV and click Edit System Environment Variables

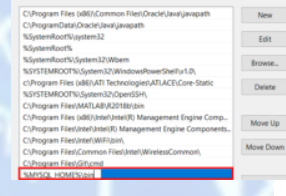
Click Environment variables
System Variables - click NEW



Click Environment variables
System Variables - click NEW
- Add MYSQL_HOME



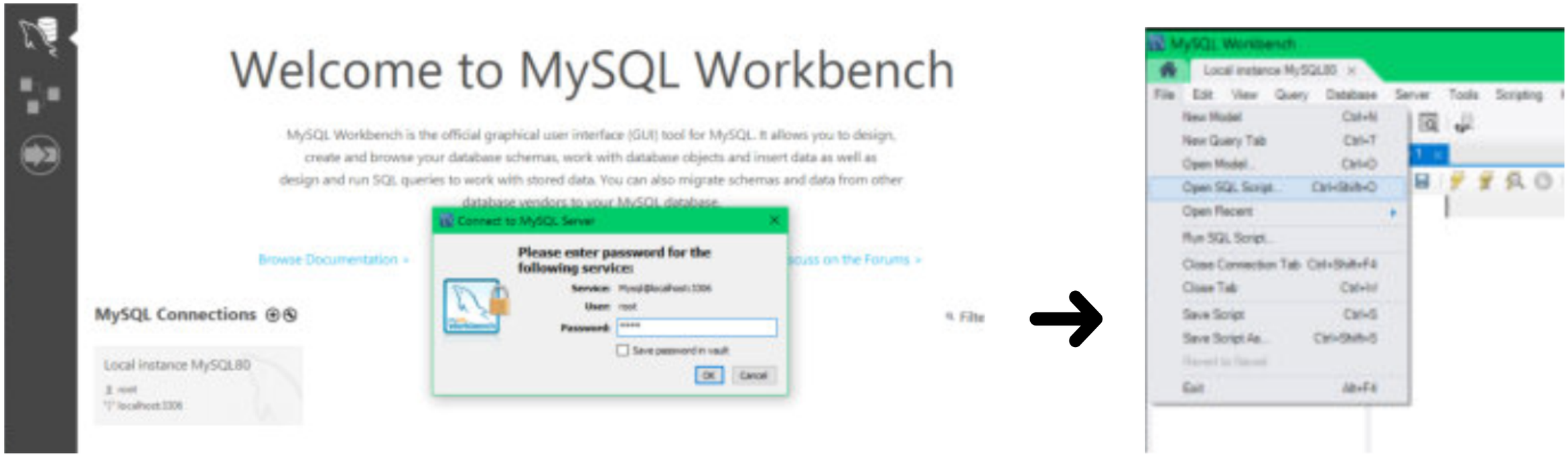
System Variables - scroll down
- Find PATH and click EDIT
- Click NEW
- Add %MYSQL_HOME%\bin
- Click ok>ok...



Sample Database

Download sample movielens.sql script

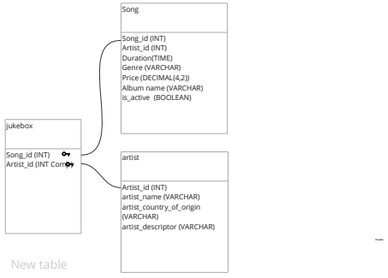
Import database into MySQL installation

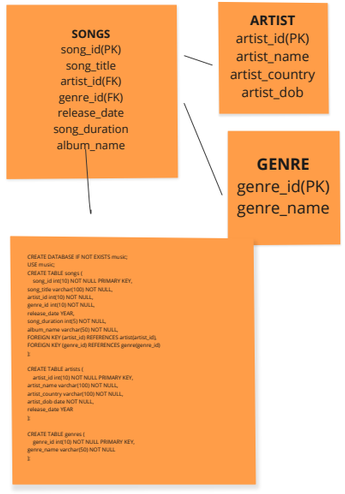


Open SQL script movielens.sql and RUN

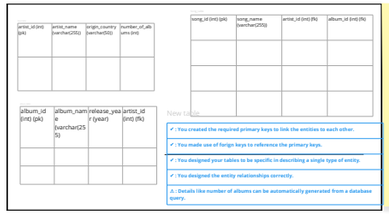
OR, open CMD prompt and run:

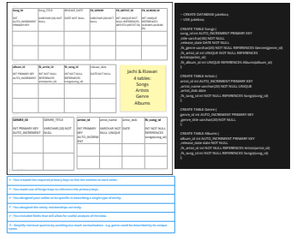
- C:\mysql -u root -p
- Enter root password 'lotr'
- mysql> SOURCE C:\<path_to_script>\movielens.sql
- mysql> SHOW TABLES FROM movielens;

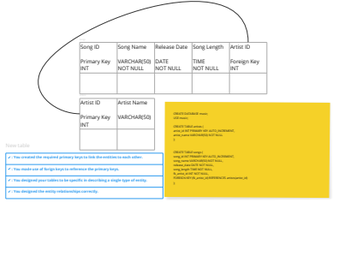




Claudia and Garvit







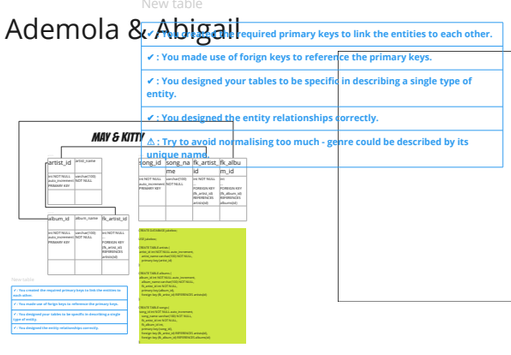
Dipraj and Calllum

```
CREATE DATABASE IF NOT EXISTS jukebox_da

USE jukebox_database;

CREATE TABLE IF NOT EXISTS artist (
artist_id int AUTO_INCREMENT,
artist_name varchar(50) NOT NULL,
dob date NOT NULL,
PRIMARY KEY(artist_id)
);

CREATE TABLE IF NOT EXISTS album (
album_id int AUTO_INCREMENT,
album_name varchar(100) NOT NULL,
num_songs int NOT NULL,
release_date date NOT NULL,
fk_artist_id int NOT NULL,
PRIMARY KEY (album_id),
FOREIGN KEY (fk_artist_id) REFERENCES artist(artist_id)
);
```

```
CREATE TABLE IF NOT EXISTS song (
song_id int AUTO_INCREMENT,
song_name varchar(50) NOT NULL,
length time NOT NULL,
release_date date NOT NULL,
fk_album_id int,
PRIMARY KEY (song_id),
FOREIGN KEY (fk_album_id) REFERENCES
album(album_id)
);
```

New table

Ademola & Abigail



New table

✔ : You created the required primary keys to link the entities to each other.

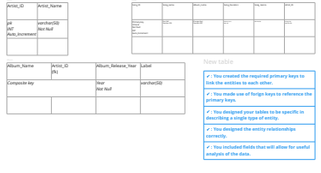
✔ : You made use of forign keys to reference the primary keys.

✔ : You designed your tables to be specific in describing a single type of entity.

✔ : You included fields that will allow for useful analysis of the data.

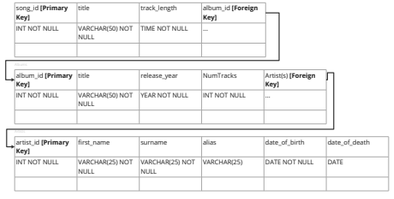
⚠ : It will be difficult to search for the songs related to the artist.

Dylan & Adeen




```
CREATE  Schema jukebox;

CREATE TABLE Artists (
  Artist_ID INT auto_increment,
  Artist_Name VARCHAR(50) NOT NULL,
  primary key (Artist_ID)
);

CREATE TABLE Albums (
  Album_Name VARCHAR(50) NOT NULL,
  Artist_ID INT NOT NULL,
  Release_Year year NOT NULL,
  Label varchar(50),
  foreign key (Artist_ID) references Artists(Artist_ID),
  primary key (Album_Name, Artist_ID)
);

CREATE TABLE Genres (
  Genre_ID INT auto_increment,
  Genre VARCHAR(50),
  Primary key (Genre_ID)
);

CREATE TABLE Songs (
  Song_ID INT auto_increment,
  Song_Title VARCHAR(50) NOT NULL,
  Genre_ID INT NOT NULL,
  Artist_ID INT NOT NULL,
  Album_Name VARCHAR(50) NOT NULL,
  foreign key (Artist_ID, Album_Name) references Albums(Artist_ID, Album_Name),
  foreign key (Genre_ID) References Genres(Genre_ID),
  primary key (Song_ID)
);
```

Ajmal and Faizal




```
/*
Music Database creation script in mysql.
Ajmal and Faizal
*/

CREATE DATABASE Jukebox;
USE Jukebox;

CREATE TABLE Artists (
  artist_id INT PRIMARY KEY AUTO_INCREMENT,
  first_name VARCHAR(25) NOT NULL DEFAULT "UNDEFINED",
  surname VARCHAR(25) NOT NULL DEFAULT "UNDEFINED",
  alias VARCHAR(25),
  date_of_birth DATE NOT NULL,
  date_of_death DATE
);

CREATE TABLE Albums (
  album_id INT PRIMARY KEY AUTO_INCREMENT,
  title VARCHAR(50) NOT NULL DEFAULT "UNDEFINED",
  release_year YEAR NOT NULL,
  NumTracks INT NOT NULL,
  artist_id INT NOT NULL,
  FOREIGN KEY (artist_id) REFERENCES Artists(artist_id)
);

CREATE TABLE Songs (
  song_id INT PRIMARY KEY AUTO_INCREMENT,
  title VARCHAR(50) NOT NULL DEFAULT "UNDEFINED",
  track_length TIME NOT NULL,
  album_id INT NOT NULL,
  FOREIGN KEY (album_id) REFERENCES Albums(album_id)
);

show tables;
select * from Artists;
```

SONGS

Songs

Song_ID	Song_name	Song_Duration	Genre_ID	Song_rating	Album_ID	Artist_ID
Primary key Unique Not Null INT Auto_Increment	Not Null Unique Varchar (50)	Decimal (4,2) Not null	Int Foreign Key references genres(genre_id)	Check (rating BETWEEN 1 AND 10) Tinyint	int FOREIGN KEY (album_id) REFERENCES albums(id)	int FOREIGN KEY (artist_id) REFERENCES artists(id)
New table						

ARTISTS

Artists

Artist_ID	Artist_Name	Artist_Email
Primary key *Unique* *Not Null* *INT* *Auto_Increment*	*varchar(50)* *Not Null*	*Unique* *Not Null* *varchar(50)*

ALBUMS

ALBUMS

album_id	album_name	artist_id
Primary key *Unique* *Not Null* *INT* *Auto_Increment*	varchar(100) NOT NULL	int NOT NULL ... FOREIGN KEY (fk_artist_id) REFERENCES artists(id)

GENRES

Artists

Genre_ID	Genre_name
Primary key *Unique* *Not Null* *INT* *Auto_Increment*	*varchar(50)* *Not Null*

✔ : You created the required primary keys to link the entities to each other.

✔ : You made use of forign keys to reference the primary keys.

✔ : You designed your tables to be specific in describing a single type of entity.

✔ : You designed the entity relationships correctly.

✔ : You included fields that will allow for useful analysis of the data.

⚠ : Simplify retrieval queries by avoiding too much normalisation - e.g. genre could be described by its unique name.

SELECT Statement - 3 Capabilities

EMPLOYEES

Emp ID	First	Last	Job-id	Email	Salary	Dept ID
101	Robert	Burns	NULL	rb@gmail	300k	10
102	Billy	Connolly	Sys_Admin	billy@gmail	55k	20
103	Rob	Roy	IT_prog	rob@gmail	100k	20
104	William	Wallace	Sales_exec	willy@gmail	NULL	30
105	Kenny	Dalglish	DB_dev	kenny@gmail	250k	40
106	Rod	Stewart	Sales_exec	rod@gmail	75k	30
107	Julia	Donaldson	Sales_exec	julia@gmail	50k	30

Projection (columns)

Emp ID	First	Last	Dept ID	Email
101	Robert	Burns	10	rb@gmail
102	Billy	Connolly	20	billy@gmail
103	Rob	Roy	20	rob@gmail
104	William	Wallace	30	willy@gmail
105	Kenny	Dalglish	40	kenny@gmail
106	Rod	Stewart	30	rod@gmail
107	Julia	Donaldson	30	julia@gmail

Returns all rows.

Selection/Filtering(rows)

Emp ID	First	Last	Job-id	Email	Salary	Dept ID
101	Robert	Burns	NULL	rb@gmail	300k	10
102	Billy	Connolly	Sys_Admin	billy@gmail	55k	20
106	Rod	Stewart	Sales_exec	rod@gmail	75k	30
107	Julia	Donaldson	Sales_exec	julia@gmail	50k	30

n x rows

```
SELECT      *
FROM        dbo.Products
WHERE       predicate/expression
ORDER BY ColA, ColB, ColC;
```

Joining rows on n x tables

CategoryName	Prod ID	ProdName	Supp ID	Cat ID	UnitPrice	SupplD	CompanyName	City
Beverages	1	Chai	1	1	18.00	1	Exotic Liquids	London
Beverages	2	Chang	1	1	19.00	1	Exotic Liquids	London
Condiments	3	Aniseed	1	2	10.00	1	Exotic Liquids	London
Condiments	4	Cajun Season	2	2	22.00	2	New Orleans	New Orleans
Condiments	5	Gumbo Mix	2	2	21.35	2	New Orleans	New Orleans
Condiments	5	Gumbo Mix	2	2	21.35	2	New Orleans	New Orleans

```
-- CREATE DATABASE jukebox;
-- USE jukebox;
-- DROP TABLE songs;
CREATE TABLE songs (
        song_id     INT PRIMARY KEY AUTO_INCREMENT
        ,title        VARCHAR(50) NOT NULL
        ,duration     DECIMAL(4,2)
        ,genre_id    INT    REFERENCES genres(genre_id)
        ,rating        TINYINT CHECK (rating BETWEEN 1 AND 10)
        ,album_id    INT REFERENCES albums(album_id)
        ,artist_id    INT REFERENCES artists(artist_id)
        ,price        DECIMAL(4,2) DEFAULT 0.59
        );

CREATE TABLE genres (
        genre_id    INT PRIMARY KEY AUTO_INCREMENT
        ,genre_name    VARCHAR(50)
        );

CREATE TABLE artists (
        artist_id    INT PRIMARY KEY AUTO_INCREMENT
        ,name        VARCHAR(50)
        ,email        VARCHAR(50) UNIQUE
        );

CREATE TABLE albums (
        album_id    INT PRIMARY KEY AUTO_INCREMENT
        ,album_name    VARCHAR(100) NOT NULL
        ,artist_id    INT REFERENCES artists(artist_id)
        );
```

```
USE jukebox;

INSERT INTO songs (title,duration,genre_id
                  ,rating,album_id,artist_id,price)
VALUES  ('killer queen', 3.50, 2, 9, NULL, 4, 0.59),
        ('mamma mia', 5.10, 1, 10, NULL, 1, 0.99),
        ('dancing queen', 5.20, 1, 9, NULL, 1, 0.59),
        ('heart of glass', 4.10, 1, 9, NULL, 6, 0.59);

SELECT * FROM songs;
```

```
USE jukebox;

-- INSERT rows using DML statements.
INSERT INTO genres (genre_name)
VALUES ('pop');

INSERT INTO genres (genre_name)
VALUES ('rock');

INSERT INTO genres (genre_name)
VALUES ('country');

INSERT INTO genres (genre_name)
VALUES ('hip hop');

INSERT INTO genres (genre_name)
VALUES ('dance');

INSERT INTO genres (genre_name)
VALUES ('emo'),
       ('scremo');

SELECT * FROM genres;
```

```
USE jukebox;

INSERT INTO artists (name, email)
VALUES ('abba','anna@gmail'),
       ('silk sonic','dave@gmail'),
       ('one republic','tim@hotmail'),
       ('queen','brian@gmail'),
       ('paramore', 'sharon@hotmail'),
       ('blondie', 'deb@gmail');

SELECT * FROM artists;
```

PROJECTION (SELECT)

```
USE jukebox;

-- Example of PROJECTION - you choose which cols to project
SELECT     song_id AS "ID"
        , title AS "Song Title"
        , rating
        , duration
        , price
        , (price * 1.10) + 0.35 AS "Brexit Price"
        , 100 % 30
        , 'hello world'
FROM    songs;
```

SORTING (ORDER BY)

```
USE jukebox;

-- Col Aliases are GENERATED after WHERE but before ORDER
BY so CAN USE col alises in
-- ORDER BY clause.
SELECT  song_id AS "ID"
  , title AS "Song Title"
        , rating
        , duration
        , price
        , (price * 1.10) + 0.35 AS BrexitPrice
        , released
FROM songs
WHERE   song_id >= 1
ORDER BY rating DESC, title ASC; -- BY col/s PREFERRED
-- ORDER BY BrexitPrice DESC -- By COL ALIAS - Prefereed!
-- ORDER BY 6 DESC -- BY col position NOT RECOMMENDED
IN CASE SELECT cols are changed
-- ORDER BY (price * 1.10) + 0.35 DESC; -- BY expression
-- ORDER BY rating DESC, title ASC; -- BY col/s PREFERRED
```

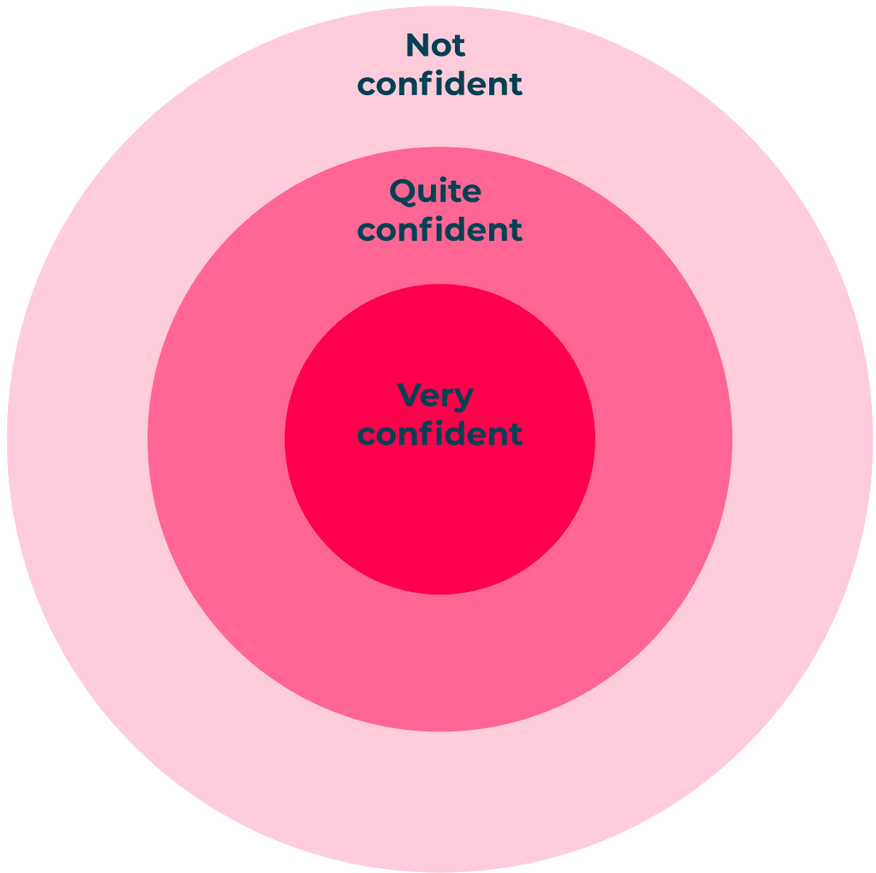
SELECTION (WHERE)

```
-- Example of SELECTION - filter rows returned.
-- Conditional operators: = != <> > >= < <= for all DATATYPES
-- Boolean Operators: NOT AND OR
-- Special Operators: [NOT] BETWEEN lo AND hi, [NOT] IN (val,
val2, val3)
-- Special Operators: [NOT] LIKE 'pattern', IS [NOT] NULL
SELECT     song_id AS "ID"
        , title AS "Song Title"
        , rating
        , duration
        , price
        , (price * 1.10) + 0.35 AS "Brexit Price"
        , released
FROM    songs
WHERE    released IS NOT NULL;

-- WHERE price < 0.6;
-- WHERE price <= 0.6;
-- WHERE price = 0.99;
-- WHERE price != 0.99;
-- WHERE price <> 0.99; ANSI SQL for not-equal-to
-- WHERE title = 'killer queen';
-- WHERE title >= 'killer queen';
-- WHERE released = '2022-10-19';
-- WHERE released <> '2022-10-19';
-- WHERE price >= 0.5 AND price =< 0.7;
-- WHERE price BETWEEN 0.5 AND 0.7;
-- WHERE title BETWEEN 'A' AND 'L';
-- WHERE released BETWEEN '2022-01-01' AND '2022-12-31';
-- WHERE price = 0.20 OR price = 0.30 OR price = 0.69;
-- WHERE price IN (0.2, 0.3, 0.69);
-- WHERE title LIKE '%k%';
-- WHERE title LIKE 'k_l%';
-- WHERE released LIKE '2022%';
-- WHERE regexp_like(title, '^[A-D]');
-- WHERE released IS NOT NULL;
```

Short Retrospective

Write something

Draw something



Grouping/Aggregation Functions

Grouping/Aggregating Functions (Summary Report)

- min(),max(),sum(),avg(),count(),count_big(),var(),stdev()..
- numeric = min/max/sum/avg/count
- string = min/max/count
- date/time = min/max/count
- NULLs are ignored!

Northwind

dbo.Products

Prod ID	ProdName	Supp ID	Cat ID	UnitPrice	UnitsInStock	UnitsOnOrder
1	Chai	1	1	18.00	39	0
2	Chang	1	1	19.00	17	40
3	Aniseed	1	2	10.00	13	70
4	Cajun Season	2	2	22.00	53	0
5	Gumbo Mix	2	2	21.35	0	0
6	Boysenberry	3	2	25.00	120	0
7	Dried Pears	3	7	30.00	15	0
8	Cranberry	3	2	40.00	6	0
9	Mishi Kobe	4	6	97.00	29	0
6	Boysenberry	3	2	25.00	120	0
7	Dried Pears	3	7	30.00	15	0
8	Cranberry	3	2	40.00	6	0
9	Mishi Kobe	4	6	97.00	29	0

Product DETAIL

```
SELECT      ProdID, ProdName
          , SupplierID,CategoryID
          , UnitPrice
FROM        dbo.Products
WHERE       ProductID < 10
ORDER BY    UnitPrice ASC
LIMIT       100
OFFSET      0;
```

Prod ID	ProdName	Supp ID	Cat ID	UnitPrice
1	Chai	1	1	18.00
2	Chang	1	1	19.00
3	Aniseed	1	2	10.00
4	Cajun Season	2	2	22.00
5	Gumbo Mix	2	2	21.35
6	Boysenberry	3	2	25.00
7	Dried Pears	3	7	30.00
8	Cranberry	3	2	40.00
9	Mishi Kobe	4	6	97.00

Product DETAIL

Grouping/Aggregation Functions

Grouping/Aggregating Functions (Summary Report)

- min(),max(),sum(),avg(),count(),count_big(),var(),stdev()..
- numeric = min/max/sum/avg/count
- string = min/max/count
- date/time = min/max/count
- NULLs are ignored!

Northwind

dbo.Products

Prod ID	ProdName	Supp ID	Cat ID	UnitPrice	UnitsInStock	UnitsOnOrder
1	Chai	1	1	18.00	39	0
2	Chang	1	1	19.00	17	40
3	Aniseed	1	2	10.00	13	70
4	Cajun Season	2	2	22.00	53	0
5	Gumbo Mix	2	2	21.35	0	0
6	Boysenberry	3	2	25.00	120	0
7	Dried Pears	3	7	30.00	15	0
8	Cranberry	3	2	40.00	6	0
9	Mishi Kobe	4	6	97.00	29	0

Product DETAIL

```
SELECT    min(UP),max(UP),sum(UP),avg(UP),count(UP)
FROM      dbo.Products
WHERE     ProductID < 10
ORDER BY  UnitPrice ASC;
```

Min	Max	Sum	Avg	Count
10.00	97.00	282.35	31.3722	9

Product Summary

Grouping/Aggregation Functions

Grouping/Aggregating Functions (Summary Report)

- min(),max(),sum(),avg(),count(),var(),stdev()..
- numeric = min/max/sum/avg/count
- string = min/max/count
- date/time = min/max/count
- NULLs are ignored!

Northwind

dbo.Products

Prod ID	ProdName	Supp ID	Cat ID	UnitPrice	UnitsInStock	UnitsOnOrder
1	Chai	1	1	18.00	39	0
2	Chang	1	1	19.00	17	40
3	Aniseed	1	2	10.00	13	70
4	Cajun Season	2	2	22.00	53	0
5	Gumbo Mix	2	2	21.35	0	0
6	Boysenberry	3	2	25.00	120	0
7	Dried Pears	3	7	30.00	15	0
8	Cranberry	3	2	40.00	6	0
9	Mishi Kobe	4	6	97.00	29	0

Product DETAIL

```
5.SELECT    SupplierID,min(UP),max(UP),sum(UP),avg(UP),count(UP)
1.FROM      dbo.Products
2.WHERE     ProductID < 10
3.GROUP BY  SupplierID,CategoryID
4.HAVING    avg(UP) > 30
6.ORDER BY  UnitPrice ASC;
7.-- LIMIT     0
8.-- OFFSET    10
```

Order of Evaluation

SupID	Min	Max	Sum	Avg	Count
1	10	19.00	47.00	15.6666	3
2	21.35	22.00	43.35	21.675	2
3	25.00	40.00	95.00	31.6666	3
4	97.00	97.00	97.00	97.00	1

Product Summary/group

Grouping and GROUP BY

```
USE jukebox;
-- SELECT * FROM songs;

SELECT genre_id AS Genre
  , artist_id AS Artist
  , min(rating) AS Min
        , max(rating) AS Max
        , sum(rating) AS Sum
        , avg(rating) AS Avg
        , count(rating) AS Count
FROM  songs
WHERE song_id BETWEEN 1 AND 10   -- Filter rows selected from SOURCE
GROUP BY genre_id, artist_id    -- Apply group functions per subset
HAVING  avg(rating) >= 9.5     -- Limit grouped rows
ORDER BY genre_id ASC, artist_id ASC;
```

Counting Rows in a table

```
-- TRUE number of rows in a table?
SELECT count(released)   -- Incorrect, CAREFUL NULLS are ignored.
  , count(song_id)  -- Good, PRIMARY keys are NOT NULL.
        , count(1)    -- Good, count num of literal values in rows!
        , count(*)    -- Best Practice, count num of rows.
FROM    songs;
```

De-normalisation or Joins

Denormalisation - Joining Tables using a 2-way join

dbo.Products AS dp

Prod ID	ProdName	Supp ID	Cat ID	UnitPrice	UnitsInStock	UnitsOnOrder
1	Chai	1	1	18.00	39	0
2	Chang	1	1	19.00	17	40
3	Aniseed	1	2	10.00	13	70
4	Cajun Season	2	2	22.00	53	0
5	Gumbo Mix	2	2	21.35	0	0
6	Boysenberry	3	2	25.00	120	0
7	Dried Pears	3	7	30.00	15	0
8	Cranberry	3	2	40.00	6	0
9	Mishi Kobe	4	6	97.00	29	0
77	Mishi Kobe	4	6	97.00	29	0
78	Manky Bread	NULL	NULL	97.00	0	0

Foreign Key

dbo.Suppliers AS ds

SuppID	CompanyName	ContactName	City	Region	Country	Fax
1	Exotic Liquids	Purchasing Mgr	London	NULL	UK	NULL
2	New Orleans	Order Admin	New Orleans	LA	USA	NULL
3	Grandma Kelly	Sales Rep	Ann Arbor	MI	USA	0666
4	Tokyo Traders	Mktg Mgr	Tokyo	NULL	Japan	NULL
5	COOP	Export Admin	Oviedo	Astunias	Spain	NULL
6	Mayumi	Mktg Rep	Osaka	NULL	Japan	NULL
7	Pavlova Ltd	Mktg Mgr	Melbourne	Victoria	Australia	0333
8	Special Biscuits	Sales Rep	Manchester	NULL	UK	NULL
29	Forets d'erables	Account Mgr	Ste-Hyacinth	Quebec	Canada	0999
30	Spanky Bakery	Bread Baron	Manchester	NULL	UK	0999

Primary Key

Rows from LHS
not included in report

78	Manky Bread	NULL	NULL	97.00

Rows from RHS
not included in report

30	Spanky Bakery	Manchester

```
SELECT      dp.ProductID
          , dp.ProductName
          , dp.SupplierID
          , dp.CategoryID
          , dp.UnitPrice
          , ds.SupplierID
          , ds.CompanyName,  ds.City
FROM        dbo.Products  AS dp
INNER JOIN  dbo.Suppliers AS ds ON dp.SupplierID = ds.SupplierID;
```

Prod ID	ProdName	Supp ID	Cat ID	UnitPrice	SuppID	CompanyName	City
1	Chai	1	1	18.00	1	Exotic Liquids	London
2	Chang	1	1	19.00	1	Exotic Liquids	London
3	Aniseed	1	2	10.00	1	Exotic Liquids	London
4	Cajun Season	2	2	22.00	2	New Orleans	New Orleans
5	Gumbo Mix	2	2	21.35	2	New Orleans	New Orleans
77	Mishi Kobe	4	6	97.00	4	Tokyo Traders	Tokyo

dbo.Products

Prod ID	ProdName	Supp ID	Cat ID	UnitPrice	UnitsInStock	UnitsOnOrder
1	Chai	1	1	18.00	39	0
2	Chang	1	1	19.00	17	40
3	Aniseed	1	2	10.00	13	70
4	Cajun Season	2	2	22.00	53	0
5	Gumbo Mix	2	2	21.35	0	0
6	Boysenberry	3	2	25.00	120	0
7	Dried Pears	3	7	30.00	15	0
8	Cranberry	3	2	40.00	6	0
9	Mishi Kobe	4	6	97.00	29	0
77	Mishi Kobe	4	6	97.00	29	0
78	Manky Bread	NULL	NULL	97.00	0	0

Foreign Key

dbo.Suppliers

SuppID	CompanyName	ContactName	City	Region	Country	Fax
1	Exotic Liquids	Purchasing Mgr	London	NULL	UK	NULL
2	New Orleans	Order Admin	New Orleans	LA	USA	NULL
3	Grandma Kelly	Sales Rep	Ann Arbor	MI	USA	0666
4	Tokyo Traders	Mktg Mgr	Tokyo	NULL	Japan	NULL
5	COOP	Export Admin	Oviedo	Astunias	Spain	NULL
6	Mayumi	Mktg Rep	Osaka	NULL	Japan	NULL
7	Pavlova Ltd	Mktg Mgr	Melbourne	Victoria	Australia	0333
8	Special Biscuits	Sales Rep	Manchester	NULL	UK	NULL
29	Forets d'erables	Account Mgr	Ste-Hyacinth	Quebec	Canada	0999
30	Spanky Bakery	Bread Baron	Manchester	NULL	UK	0999

Primary Key

```
SELECT      dp.ProductID
            , dp.ProductName
            , dp.SupplierID
            , dp.CategoryID
            , dp.UnitPrice
            , ds.SupplierID
            , ds.CompanyName,  ds.City
FROM        dbo.Products  AS dp
FULL OUTER JOIN   dbo.Suppliers AS ds ON dp.SupplierID = ds.SupplierID;
```

Prod ID	ProdName	Supp ID	Cat ID	UnitPrice	SuppID	CompanyName	City
1	Chai	1	1	18.00	1	Exotic Liquids	London
2	Chang	1	1	19.00	1	Exotic Liquids	London
3	Aniseed	1	2	10.00	1	Exotic Liquids	London
4	Cajun Season	2	2	22.00	2	New Orleans	New Orleans
5	Gumbo Mix	2	2	21.35	2	New Orleans	New Orleans
78	Manky Bread	NULL	NULL	97.00	NULL	NULL	NULL
NULL	NULL	NULL	NULL	NULL	30	Spanky Bakery	Manchester

LEFT OUTER JOIN

Rows from LHS

NOW included in report

FULL OUTER JOIN

RIGHT OUTER JOIN

Rows from RHS

NOW included in report

NAME THE VIDEO GAME CHARACTER?

Add some post its!

Press N to create your own sticky notes

p

Hello

Hiiiii

hey

HELLO!

Hi!

Hi

hi

Hi

Draw something

Press P to create yourfart

bear

Write something

Press T to express your thoughts

hello

Bit peckish

hiya

hiya

Hello

hi

Example of a 3-way Join

dbo.Categories AS dc

CatID	CategoryName	Description	Picture
1	Beverages	Soft drinks, coffees	0x151
2	Condiments	Sweeat & Savoury sauces	0x151
3	Confections	Desserts, candies	0x151
4	Dairy Products	Cheeses	0x151
5	Grains/Cereals	Breads, crackers	0x151
6	Meat/Poultry	Prepared meats	0x151
7	Produce	Dried Fruit	0x151
8	Seafood	Seaweed & fish	0x151

dbo.Products AS dp

Prod ID	ProdName	Supp ID	Cat ID	UnitPrice	UnitsInStock	UnitsOnOrder
1	Chai	1	1	18.00	39	0
2	Chang	1	1	19.00	17	40
3	Aniseed	1	2	10.00	13	70
4	Cajun Season	2	2	22.00	53	0
5	Gumbo Mix	2	2	21.35	0	0
6	Boysenberry	3	2	25.00	120	0
7	Dried Pears	3	7	30.00	15	0
8	Cranberry	3	2	40.00	6	0
9	Mishi Kobe	4	6	97.00	29	0
77	Mishi Kobe	4	6	97.00	29	0
78	Manky Bread	NULL	NULL	97.00	0	0

dbo.Suppliers AS ds

SuppID	CompanyName	ContactName	City	Region	Country	Fax
1	Exotic Liquids	Purchasing Mgr	London	NULL	UK	NULL
2	New Orleans	Order Admin	New Orleans	LA	USA	NULL
3	Grandma Kelly	Sales Rep	Ann Arbor	MI	USA	0666
4	Tokyo Traders	Mktg Mgr	Tokyo	NULL	Japan	NULL
5	COOP	Export Admin	Oviedo	Astunias	Spain	NULL
6	Mayumi	Mktg Rep	Osaka	NULL	Japan	NULL
7	Pavlova Ltd	Mktg Mgr	Melbourne	Victoria	Australia	0333
8	Special Biscuits	Sales Rep	Manchester	NULL	UK	NULL
29	Forets d'erables	Account Mgr	Ste-Hyacinth	Quebec	Canada	0999
30	Spanky Bakery	Bread Baron	Manchester	NULL	UK	0999

```
SELECT      dp.ProductID
          , dp.ProductName
          , dp.SupplierID
          , dp.CategoryID
          , dp.UnitPrice
          , ds.CompanyName,  ds.City, dc.CategoryName
FROM        dbo.Products    AS dp
INNER JOIN  dbo.Suppliers   AS ds ON dp.SupplierID = ds.SupplierID
INNER JOIN  dbo.categories AS dc ON dp.CategoryID = dc.CategoryID;
```

CategoryName	Prod ID	ProdName	Supp ID	Cat ID	UnitPrice	SuppID	CompanyName	City
Beverages	1	Chai	1	1	18.00	1	Exotic Liquids	London
Beverages	2	Chang	1	1	19.00	1	Exotic Liquids	London
Condiments	3	Aniseed	1	2	10.00	1	Exotic Liquids	London
Condiments	4	Cajun Season	2	2	22.00	2	New Orleans	New Orleans
Condiments	5	Gumbo Mix	2	2	21.35	2	New Orleans	New Orleans
Condiments	5	Gumbo Mix	2	2	21.35	2	New Orleans	New Orleans

Example of a self Join

dbo.Employees AS emp

EmployeeID	LastName	FirstName	Title	BirthDate	HireDate	City	ReportsTo
1	Davalio	Nancy	Sales Rep	1948-12-08	1992-05-01	Seattle	2
2	Fuller	Andrew	VP Sales	1952-02-19	1992-08-14	Tacoma	NULL
3	Leverling	Janet	Sales Rep	1963-08-30	1992-04-01	KirkLand	2
4	Peacock	Margaret	Sales Rep	1997-09-19	1993-05-03	Redmond	2
5	Buchanan	Steven	Sales Mgr	1995-03-04	1993-10-17	London	2
6	Suyuma	Michael	Sales Rep	1963-07-02	1993-10-17	London	5
7	King	Robert	Sales Rep	1960-05-29	1994-01-02	London	5
8	Callahan	Laura	Sales Coord	1950-01-09	1994-03-05	Seattle	2
9	Dodsworth	Anne	Sales Rep	1966-01-27	1994-11-15	London	5

Foreign Key

dbo.Employees AS mgr

EmployeeID	LastName	FirstName	Title	BirthDate	HireDate	City	ReportsTo
1	Davalio	Nancy	Sales Rep	1948-12-08	1992-05-01	Seattle	2
2	Fuller	Andrew	VP Sales	1952-02-19	1992-08-14	Tacoma	NULL
3	Leverling	Janet	Sales Rep	1963-08-30	1992-04-01	KirkLand	2
4	Peacock	Margaret	Sales Rep	1997-09-19	1993-05-03	Redmond	2
5	Buchanan	Steven	Sales Mgr	1995-03-04	1993-10-17	London	2
6	Suyuma	Michael	Sales Rep	1963-07-02	1993-10-17	London	5
7	King	Robert	Sales Rep	1960-05-29	1994-01-02	London	5
8	Callahan	Laura	Sales Coord	1950-01-09	1994-03-05	Seattle	2
9	Dodsworth	Anne	Sales Rep	1966-01-27	1994-11-15	London	5

Primary Key

Rows from LHS
not included in report

```
SELECT      concat(emp.FirstName, space(1), emp.LastName)     AS [Employee]
            , ' Reports To'                                   AS [Relationship]
            ' concat(mgr.FirstName, space(1), mgr.LastName)   AS [Manager]
FROM        dbo.Employees AS emp
INNER JOIN  dbo.Employees AS mgr  ON emp.ReportsTo = mgr.EmployeeID;
```

Employee	Relationship	Manager
Nancy Davalio	Reports To	Andrew Fuller
Janet Leverling	Reports To	Andrew Fuller
Margaret Peacock	Reports To	Andrew Fuller
Steven Buchanan	Reports To	Andrew Fuller
Michael Suyuma	Reports To	Steven Buchanan
Robert King	Reports To	Steven Buchanan
Laura Callahan	Reports To	Andrew Fuller
Anne Dowsworth	Reports To	Steven Buchanan
Andre Fuller	Reports To	

LEFT OUTER JOIN
to include excluded row

String Formatting

```
#! /usr/bin/env python3
# Name: py
# Author:    Donald Cameron
# Version :   v1.0
# Description: This program will demo different ways of formatting strings.
"""
  DocString:
"""
# Create a dict of planet info: names + distance to sun in Giga-metres.
planets = {'Mercury': 57.91,
           'Venus': 108.2,
           'Earth': 149.597870,
           'Mars': 227.94
}

# ITERATE through planets and display planet info
# using str concatenation + escape chars - UGLY!
for planet in planets:
    print("\t\t" + planet + ": " + str(planets[planet]) + " Gm " + str(hex(0xff)))

print("-" * 40)

# using str justification methods + concatenation - OK!
for planet in planets:
    print(planet.rjust(12) + ": " + str(planets[planet]).center(12, ' ') + " Gm " +
          str(hex(0xff)).rjust(6))

print("-" * 40)
# using str .format() method - GOOD!
for planet in planets:
    print("{0:>12s}: {1:^12.3f} Gm {2:>#6x}".format(planet, planets[planet], 0xff))

print("-" * 40)
# using f-strings, introduced from Py 3.5 onwards - BEST!
for planet in planets:
    print(f"{planet:>12s}: {planets[planet]:^12.3f} Gm {0xff:>#6x}")

print("-" * 40)
# Python 2 way of formatting strings! Use f-strings instead!
for planet in planets:
    print("%12s: %12.3f Gm %#6x" % (planet, planets[planet], 0xff))
```

```
#! /usr/bin/env python3
# Name: py
# Author:    Donald Cameron
# Version :   v1.0
# Description: This program will demo how to split and rejoin strings
# using the .split() and .join() str methods.
"""
  DocString:
"""

# Sample from /etc/passwd on Linux for the root user login.
line = "root:x:0:0:The Super User:/root:/bin/kshfsn"
# But I want to make changes to the string (immutable=ReadOnly!)

fields = line.split(":") # Returns a list (mutable) of objects.
fields[4] = "The Administrator"
fields[6] = "/bin/bash"
line = ":".join(fields) # Returns a str.
print("Modified str=", line)
```

```
#! /usr/bin/env python3
# Name: py
# Author:    Donald Cameron
# Version :   v1.0
# Description: This program will display the entire Unicode
# charset.
"""
  DocString:
"""

# ITERATE through all the char positions in the unicode table (0-65536)
for pos in range(0, 65536):
    try:
        print(chr(pos), end=" ")
        if pos % 16 == 0:
            print("")
    except UnicodeEncodeError:
        print(" ")
```

```
master_pin = "0123"
pin = None

while pin != master_pin:
    pin = input("Enter PIN: ")
    if pin == master_pin:
        print("Valid PIN")
    else:
        print("Invalid PIN")

print("Done")
```

Collections

```
#! /usr/bin/env python3
# Name: py
# Author:    Donald Cameron
# Version :   v1.0
# Description: This program will demo how to create, grow and shrink
# dict (Unordered Mutable Collections with Unique Keys). From Py3.6 onwards
# dict are in INSERTION order.
"""
  DocString:
"""
import pprint

# A multi-dimensional dict of lists!
movies = {'sess': ['miss congeniality', 'lion king', 'the incredibles'],
          'kaitlin': ['clueless', 'mean girls', 'lego movie'],
          'colette': ['the pianist', 'rush', 'dark knight']
}
# Add new key=objects to dict.
movies['donald'] = ['kon', 'life or brian', 'holy grail']
pprint.pprint(movies)
print("-" * 60)

print(f"Colette's favourite movies = {movies['colette']}")
print(f"Colette's favourite movies = {movies.get('colette')}")
print(f"Colette's ultimate movie = {movies['colette'][0]}")

# films = movies.copy() # Copy dict.
# films.clear() # Empty dict.
# movies.pop('donald') # Remove key=object by name.
# movies.popitem() # Removes last inserted object.

# Iterate through keys, values and both using an ITERATOR for loop.
# Iterate through keys.
for name in movies.keys():
    print(f"{name} likes {movies[name]}")

print("-" * 60)
# Iterate through values.
for films in movies.values():
    print(f"{films}")

print("-" * 60)
# Iterate through keys+values.
for name, films in movies.items():
    print(f"{name} loves {films}")
```

```
#! /usr/bin/env python3
# Name: py
# Author:    Donald Cameron
# Version :   v1.0
# Description: This program will demo how to create, grow, shrink and
# combine sets (Unordered Mutable Collections) using SET operators
# (Remember VENN diagrams!).
"""
  DocString:
"""
# Create two SETS.
marvel_fans = {'holly', 'jess', 'sam', 'shazzad', 'donald', 'colette'}
dc_fans = set() # Create an empty set

# Grow a set!
dc_fans.add('donald')
dc_fans.add('seonaid')
dc_fans.add('valle')

# dc_fans.pop() # Randomly remove an object
# comic_fans = dc_fans.copy() # Copy set.
# comic_fans.clear() # Empty set.

print(f"Fans of Marvel = {marvel_fans}")
print(f"Fans of DC = {dc_fans}")

print("-" * 60)

# Combine SETS using SET Methods.
print(f"Fans of either Marvel OR DC = {marvel_fans.union(dc_fans)}")
print(f"Fans of both Marvel AND DC = {marvel_fans.intersection(dc_fans)}")
print(f"Fans of only Marvel = {marvel_fans.difference(dc_fans)}")
print(f"Fans of only Marvel OR DC =
{marvel_fans.symmetric_difference(dc_fans)}")
print("-" * 60)
# Combine SETS using SET Operators.
print(f"Fans of either Marvel OR DC = {marvel_fans | dc_fans}")
print(f"Fans of both Marvel AND DC = {marvel_fans & dc_fans}")
print(f"Fans of only Marvel = {marvel_fans - dc_fans}")
print(f"Fans of only Marvel OR DC = {marvel_fans ^ dc_fans}")
```

Regex

```
#! /usr/bin/env python3
# Name: py
# Author:    Donald Cameron
# Version :   v1.0
# Description: This program will demo how to match string data using str testing
# and PATTERN matching using re.py module and Regex patterns.
"""
  DocString:
"""
import re
# Open file handle for READING in TEXT mode!
fh_in = open("C:\\labs\\words", mode="rt")

# Iterate through file handle (ITERATOR) one line a time.
for line in fh_in:
    # Example of str testing.
    # if line.endswith("t") and line.rstrip("\n").endswith("n") and "town" in line:
    # m = re.search("^the", line) # Match lines starting with 'the'.
    # m = re.search("ing$", line) # Match lines ending with 'ing'.
    # m = re.search("^ring$", line) # Match line only containing 'ring'.
    # m = re.search("^.{18}$", line) # Match lines with exactly 19 chars.
    # m = re.search("^.{1}$|^.{19}$", line) # Match lines with exactly 19 chars.
    # m = re.search("^[A-Z]", line) # Match lines starting with a capital.
    # m = re.search("[0-9][0-9]", line) # Match lines with 2 consecutive digits.
    # m = re.search("[aeiou]{3consecutive}", line) # Match 3 consecutive vowels.
    # m = re.search("([aeiou]){3}", line) # Match lines with at least 5 consecutive vowels.
    # m = re.search("^.$", line) # Match lines with a 'dot'.
    # m = re.search("^[A-Z].*[A-Z]$", line) # Match lines start/end with a capital.
    m = re.search("^(.).\1.$", line, flags=re.IGNORECASE) # Match lines with 5 char palindromes!
    # m = re.search("^([A-Z]).*\1$", line) # Match lines start/end with SAME capital!
    # m = re.match("^([A-Z]).*\1$", line) # AUTOMATICALLY MATCHES lines starting with pattern.
    # m = re.fullmatch("^([A-Z]).*\1$", line) # Must MATCH ENTIRE line including whitespace!
    # m = re.search("^[A-Z]{1,2}[0-9]{1,2}[A-Z]{3}$", line) # Match UK Postcode
    if m:
        print("Matched {m.group()} on {line.rstrip()} at pos {m.start()}-{m.end()}"
              f" Groupings={m.groups()}, group 1={m.group(1)}")

fh_in.close()
```

```
#! /usr/bin/env python3
# Name: py
# Author:    Donald Cameron
# Version :   v1.0
# Description: This program will demo how to match string data using str testing
# and PATTERN matching using re.py module and Regex patterns.
"""
  DocString:
"""
import re
# Open file handle for READING in TEXT mode!
fh_in = open("C:\\labs\\words", mode="rt")

re.obj = re.compile("^([A-Z]).*\1$") # Pre-compile PATTERN JUST ONCE!

# Iterate through file handle (ITERATOR) one line a time.
for line in fh_in:
    # m = re.search("^([A-Z]).*\1$", line) # Re-compile PATTERN for every line! OUCH!
    m = re.obj.search(line)
    if m:
        print(line, end="")

fh_in.close()
```

```
#! /usr/bin/env python3
# Name: py
# Author:    Donald Cameron
# Version :   v1.0
# Description: This program will demo how to match and make
# a substitution using Regex and the re.sub() function.
"""
  DocString:
"""
import re

# Sample from /etc/passwd on Linux for the root user login.
line = "root:x:0:0:The Super User:/root:/bin/kshfsn"

line = re.sub(r":[^:]+$", r":/Administrator", line) # Returns Modified str.
line = re.sub("ksh$", "bash", line) # Returns Modified TUPLE (str,num of changes)
(line, num) = re.subn("root", r"groot", line) # Returns Modified TUPLE (str,num of changes)

print("New line={line.rstrip()}, modified {num} times")
```

User Functions

```
#! /usr/bin/env python3
# Name: py
# Author:    Donald Cameron
# Version :   v1.0
# Description: This program will demo how to define, name, and call
# a user function, optionally passing parameters and returning data.
"""
  DocString:
"""
# Example of a user function with parameter passing and default values.
# Annotations - embedded comment used to describe preferred type for
# parameters (Not enforced).
def say_hello(name="hallo", recipient:str="meine freunde")->None:
    message = f"{greeting} {recipient}"
    print(message)
    return None


say_hello("hola", "mis amigos") # Positional parameter passing.
say_hello(greeting="bonjour", recipient="mes amis") # Named parameter passing.
say_hello(recipient="chinguadeu", greeting="annyeonghaseyo") # Named parameter (in any order)
say_hello("konichiwa", recipient="tomodachi") # Mixed parameter passing (positional->named)
say_hello("hallo", "mein Fruende")
say_hello()

# Can find out what anotations a user function uses!
print(f"Annotations for say_hallo is {say_hello.__annotations__}")
```

```
#! /usr/bin/env python3
# Name: py
# Author:    Donald Cameron
# Version :   v1.0
# Description: This program will demo a user function with parameter passing.
"""
  A Script to search for Regex Patterns in a file.
"""
import sys
import re

# Example of USEFUL user function with parameter passing and
# default values, and annotations.
def search_pattern(pattern:str="^([A-Z]).*\1$", file:str="c:\\labs\\words")->None:
    """ Search for Regex pattern in a file """
    # Open file handle for READING in TEXT mode!
    with open(file, mode="rt") as fh_in:
        for line in fh_in:
            m = re.search(pattern, line) # Match lines with 5 char palindromes!
            if m:
                print(f"Matched {m.group()} on {line.rstrip()} at pos {m.start()}-{m.end()}")
    return None

# Example of a VARIADIC function, which accepts variable number
# or parameters into a TUPLE!
def search_files(pattern, *files):
    """ Search for Regex pattern in multiple files """
    lines = 0
    for file in files:
        with open(file, mode="rt") as fh_in:
            for line in fh_in:
                m = re.search(pattern, line) # Match lines with 5 char palindromes!
                if m:
                    lines += 1
                    print(f"Matched {m.group()} on {line.rstrip()} at pos {m.start()}-{m.end()}")
    return lines


def main():
    search_pattern(r"^.{10}$", r"c:\labs\words")
    num = search_files(r"^.{20}$", r"c:\labs\words", r"c:\labs\words2", r"c:\labs\words3")
    print(f"Matched {num} lines")
    return None

# Namespace Trick.
if __name__ == "__main__":
    # Execute only if ran directly as a program.
    # Ignore if imported as a module
    main()
    sys.exit(0)
```

t ways of formatting stri

e to sun in Giga-metres.