SQL fundamentals

Summer School 2021

Lector: Sergey Vichev

Plan of attack

Introduction:

- What is SQL
- Dataset overview
- IDE

SQL Fundamentals (SELECT, WHERE, etc.)

SQL Intermidiate (GROUP BY, HAVING, etc)

Introduction

- **What is SQL**
- Dataset Overview
- SQLite Online IDE

YES I DO HAVE A SPECIAL SKILL SET...
WHAT? NO... WHAT THE HELL IS SQL?

SQL FUNDAMENTALS: What is SQL?

SQL is **Structured Query Language**, which is a computer language for storing, manipulating and retrieving data stored in a relational database.

SQL is the standard language for Relational Database System.

SQL FUNDAMENTALS: What is SQL?

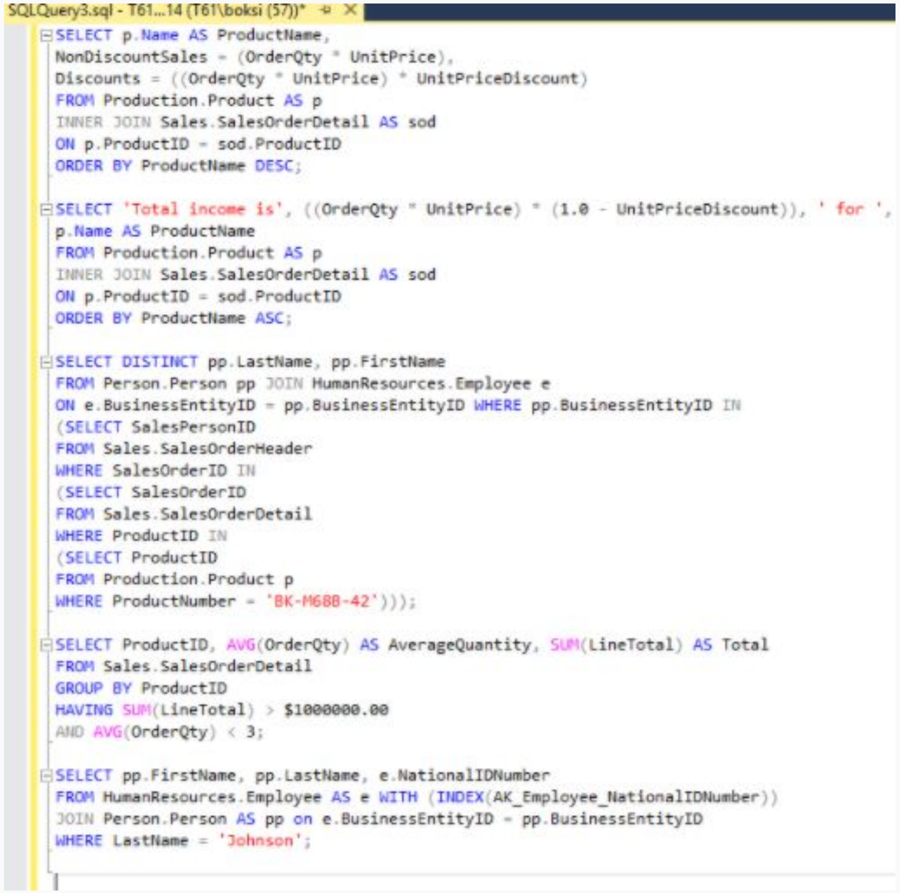
>> All the Relational Database Management Systems (RDMS) like MySQL, MS Access, Oracle, Sybase, Informix, Postgres and SQL Server use SQL as their standard database language.

Also, they are using different dialects, such as

- MS SQL Server using T-SQL,
- Oracle using PL/SQL,
- MS Access version of SQL is called JET SQL (native format) etc.

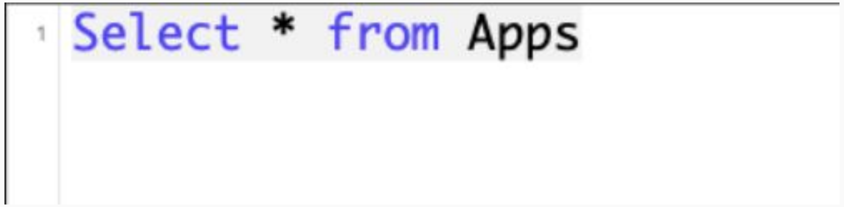
>> Versions

SQL FUNDAMENTALS: SQL Query Examples



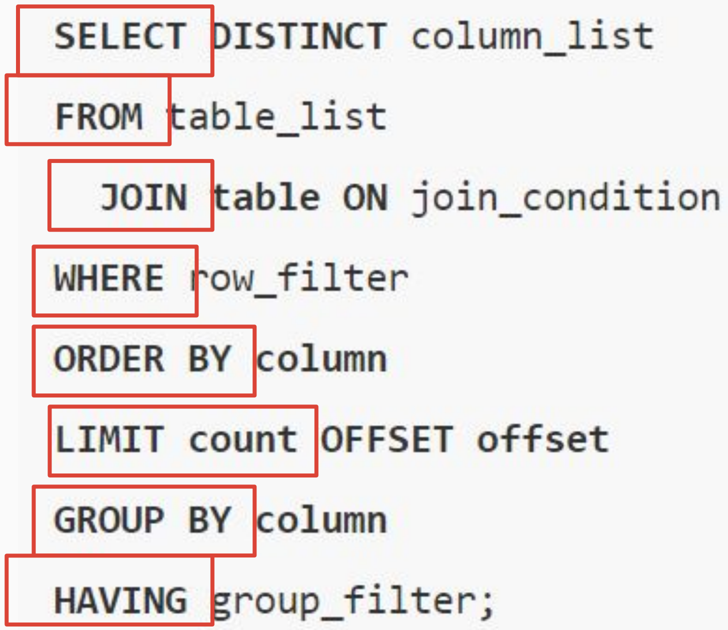
SQLQuery3.sql - T61...14 (T61\boksi (57))*

```
SELECT p.Name AS ProductName,
NonDiscountSales = (OrderQty * UnitPrice),
Discounts = ((OrderQty * UnitPrice) * UnitPriceDiscount)
FROM Production.Product AS p
INNER JOIN Sales.SalesOrderDetail AS sod
ON p.ProductID = sod.ProductID
ORDER BY ProductName DESC;

SELECT 'Total income is', ((OrderQty * UnitPrice) * (1.0 - UnitPriceDiscount)), ' for ',
p.Name AS ProductName
FROM Production.Product AS p
INNER JOIN Sales.SalesOrderDetail AS sod
ON p.ProductID = sod.ProductID
ORDER BY ProductName ASC;

SELECT DISTINCT pp.LastName, pp.FirstName
FROM Person.Person pp JOIN HumanResources.Employee e
ON e.BusinessEntityID = pp.BusinessEntityID WHERE pp.BusinessEntityID IN
(SELECT SalesPersonID
FROM Sales.SalesOrderHeader
WHERE SalesOrderID IN
(SELECT SalesOrderID
FROM Sales.SalesOrderDetail
WHERE ProductID IN
(SELECT ProductID
FROM Production.Product p
WHERE ProductNumber = 'BK-M68B-42')));

SELECT ProductID, AVG(OrderQty) AS AverageQuantity, SUM(LineTotal) AS Total
FROM Sales.SalesOrderDetail
GROUP BY ProductID
HAVING SUM(LineTotal) > $1000000.00
AND AVG(OrderQty) < 3;

SELECT pp.FirstName, pp.LastName, e.NationalIDNumber
FROM HumanResources.Employee AS e WITH (INDEX(AK_Employee_NationalIDNumber))
JOIN Person.Person AS pp on e.BusinessEntityID = pp.BusinessEntityID
WHERE LastName = 'Johnson';
```



```
Select * from Apps
```

SQL FUNDAMENTALS: SQL query structure



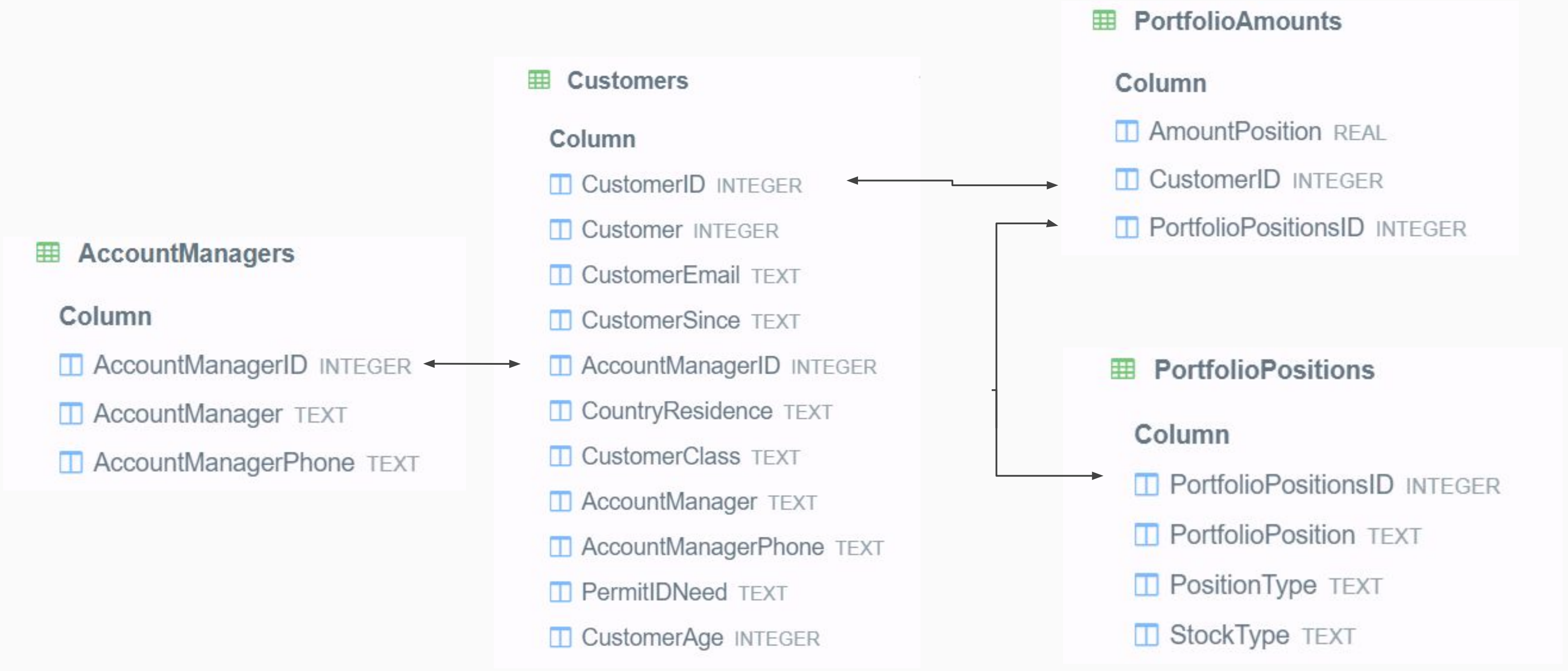
```
SELECT DISTINCT column_list
FROM table_list
  JOIN table ON join_condition
WHERE row_filter
ORDER BY column
LIMIT count OFFSET offset
GROUP BY column
HAVING group_filter;
```

- SELECT statement example is used to extract data
- Clauses: JOIN, WHERE, ORDER BY, LIMIT, GROUP BY, HAVING, etc.
- Good Practice is each clause starts from new line

Introduction

- What is SQL
- **Dataset Overview**
- SQLite Online IDE

DATASET OVERVIEW: BROKERAGE DATASET SCHEMA



AccountManagers

Column	
AccountManagerID	INTEGER
AccountManager	TEXT
AccountManagerPhone	TEXT

Customers

Column	
CustomerID	INTEGER
Customer	INTEGER
CustomerEmail	TEXT
CustomerSince	TEXT
AccountManagerID	INTEGER
CountryResidence	TEXT
CustomerClass	TEXT
AccountManager	TEXT
AccountManagerPhone	TEXT
PermitIDNeed	TEXT
CustomerAge	INTEGER

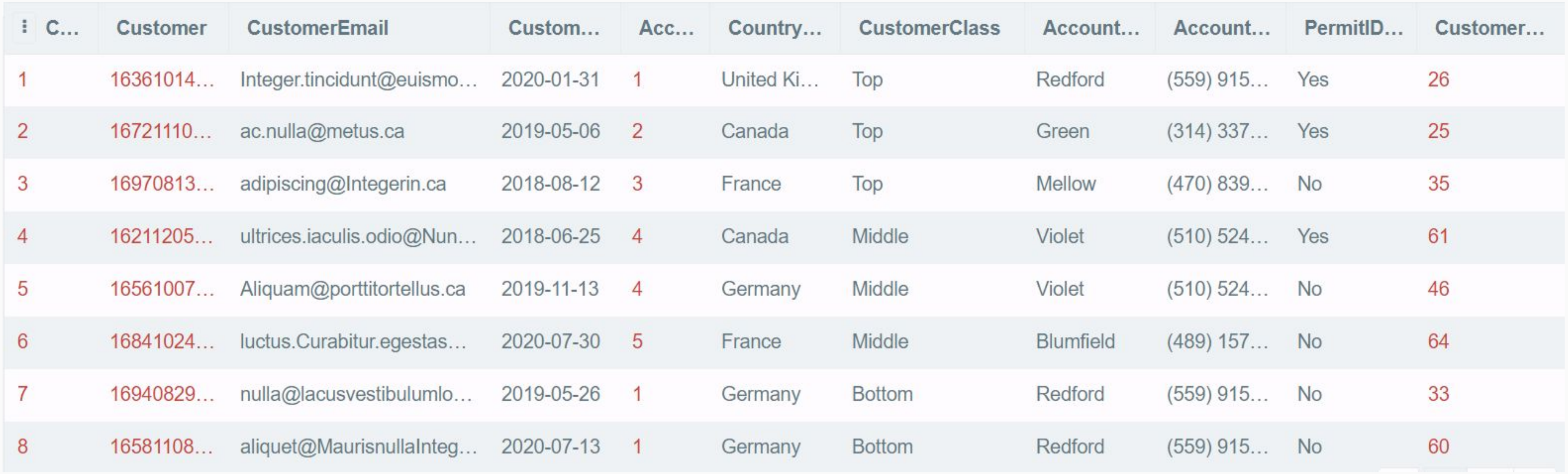
PortfolioAmounts

Column	
AmountPosition	REAL
CustomerID	INTEGER
PortfolioPositionsID	INTEGER

PortfolioPositions

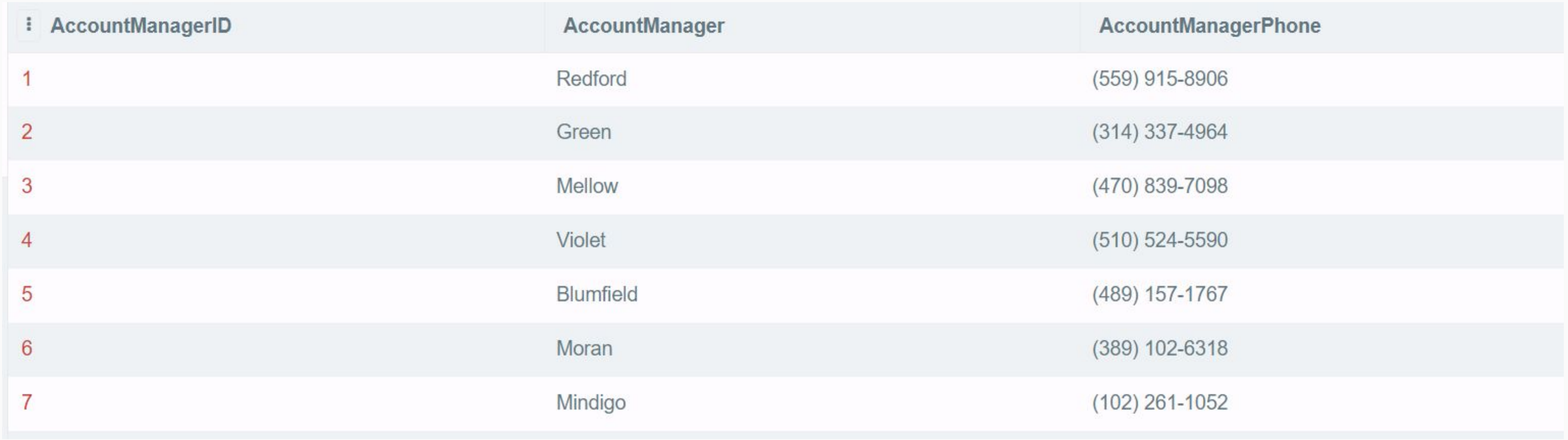
Column	
PortfolioPositionsID	INTEGER
PortfolioPosition	TEXT
PositionType	TEXT
StockType	TEXT

DATASET OVERVIEW: Customers



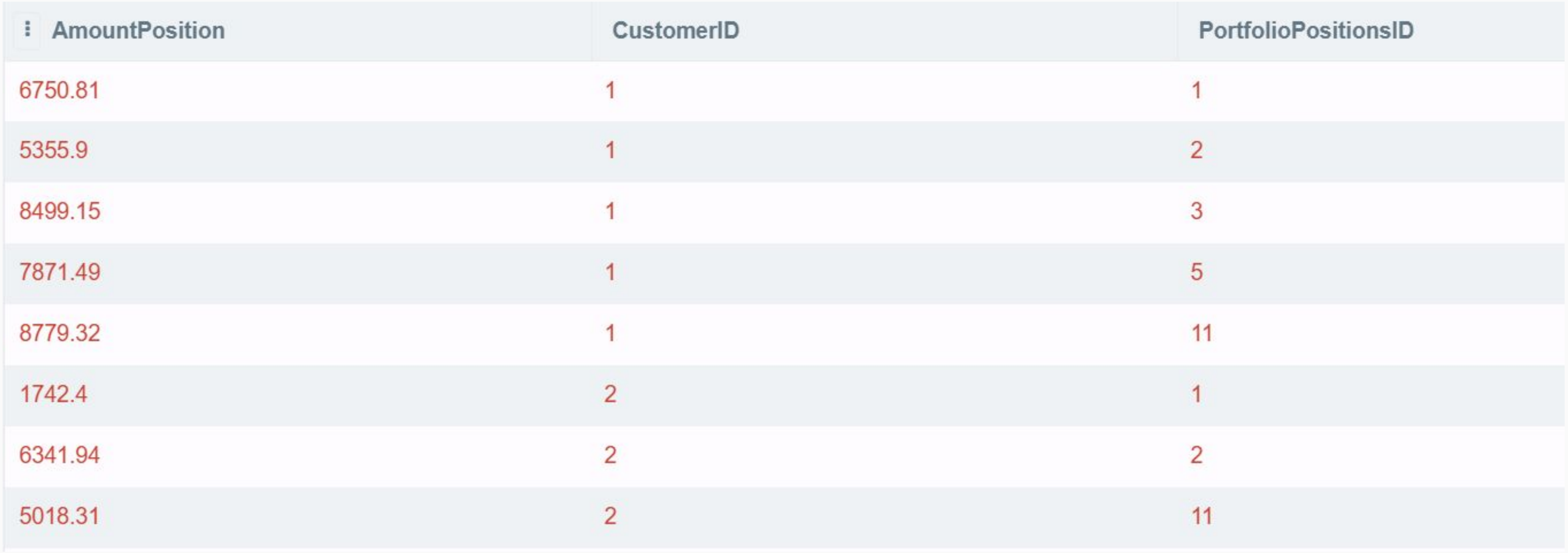
⋮ C…	Customer	CustomerEmail	Custom…	Acc…	Country…	CustomerClass	Account…	Account…	PermitID…	Customer…
1	16361014…	Integer.tincidunt@euismo…	2020-01-31	1	United Ki…	Top	Redford	(559) 915…	Yes	26
2	16721110…	ac.nulla@metus.ca	2019-05-06	2	Canada	Top	Green	(314) 337…	Yes	25
3	16970813…	adipiscing@Integerin.ca	2018-08-12	3	France	Top	Mellow	(470) 839…	No	35
4	16211205…	ultrices.iaculis.odio@Nun…	2018-06-25	4	Canada	Middle	Violet	(510) 524…	Yes	61
5	16561007…	Aliquam@porttitortellus.ca	2019-11-13	4	Germany	Middle	Violet	(510) 524…	No	46
6	16841024…	luctus.Curabitur.egestas…	2020-07-30	5	France	Middle	Blumfield	(489) 157…	No	64
7	16940829…	nulla@lacusvestibulumlo…	2019-05-26	1	Germany	Bottom	Redford	(559) 915…	No	33
8	16581108…	aliquet@MaurisnullaInteg…	2020-07-13	1	Germany	Bottom	Redford	(559) 915…	No	60

DATASET OVERVIEW: Account Managers



AccountManagerID	AccountManager	AccountManagerPhone
1	Redford	(559) 915-8906
2	Green	(314) 337-4964
3	Mellow	(470) 839-7098
4	Violet	(510) 524-5590
5	Blumfield	(489) 157-1767
6	Moran	(389) 102-6318
7	Mindigo	(102) 261-1052

DATASET OVERVIEW: Portfolio Amounts



AmountPosition	CustomerID	PortfolioPositionsID
6750.81	1	1
5355.9	1	2
8499.15	1	3
7871.49	1	5
8779.32	1	11
1742.4	2	1
6341.94	2	2
5018.31	2	11

Introduction

- What is SQL
- Dataset Overview
- **SQLite Online IDE**

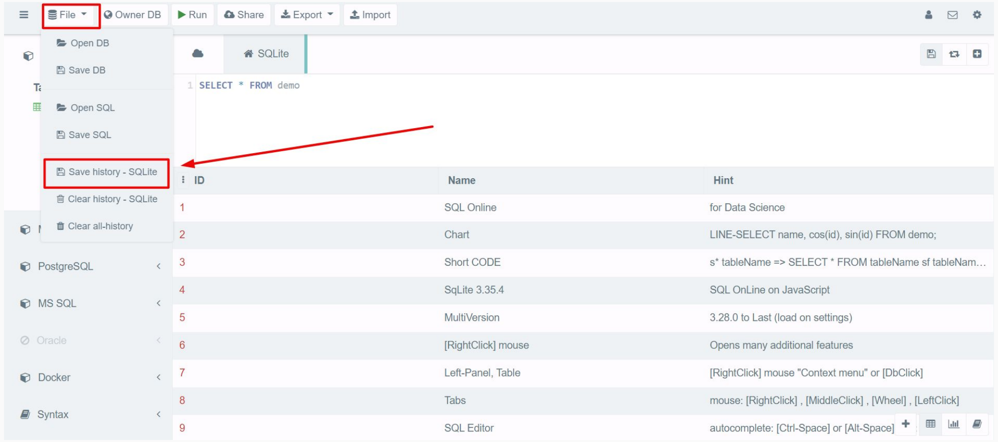
SQLite Online IDE Overview

- SQLite section
    - Tables and columns
- Overview of the tables
- Code field
    - New tabs
    - Auto-suggestion, copy to IDE
    - Run
- Query Result View
    - Column Widths
- History
- Export CSV

SQL Online IDE

ARCHIVE YOUR DATA

File >> Save History - SQL Lite



```
SELECT * FROM demo
```

ID	Name	Hint
1	SQL Online	for Data Science
2	Chart	LINE-SELECT name, cos(id), sin(id) FROM demo;
3	Short CODE	s* tableName => SELECT * FROM tableName sf tableNam…
4	SqLite 3.35.4	SQL OnLine on JavaScript
5	MultiVersion	3.28.0 to Last (load on settings)
6	[RightClick] mouse	Opens many additional features
7	Left-Panel, Table	[RightClick] mouse "Context menu" or [DbClick]
8	Tabs	mouse: [RightClick] , [MiddleClick] , [Wheel] , [LeftClick]
9	SQL Editor	autocomplete: [Ctrl-Space] or [Alt-Space]

PRACTICE: SQLite IDE

- Review tables (test the double click to open each table)
- Try to open new code tabs
- Try to save individual code tab
- Try to export CSV
- Try to save history

Dataset Link:

http://bit.ly/sql_course_su

SQL Fundamentals

- **SELECT statement**
- ORDER BY, LIMIT clauses
- WHERE and Logical Operators

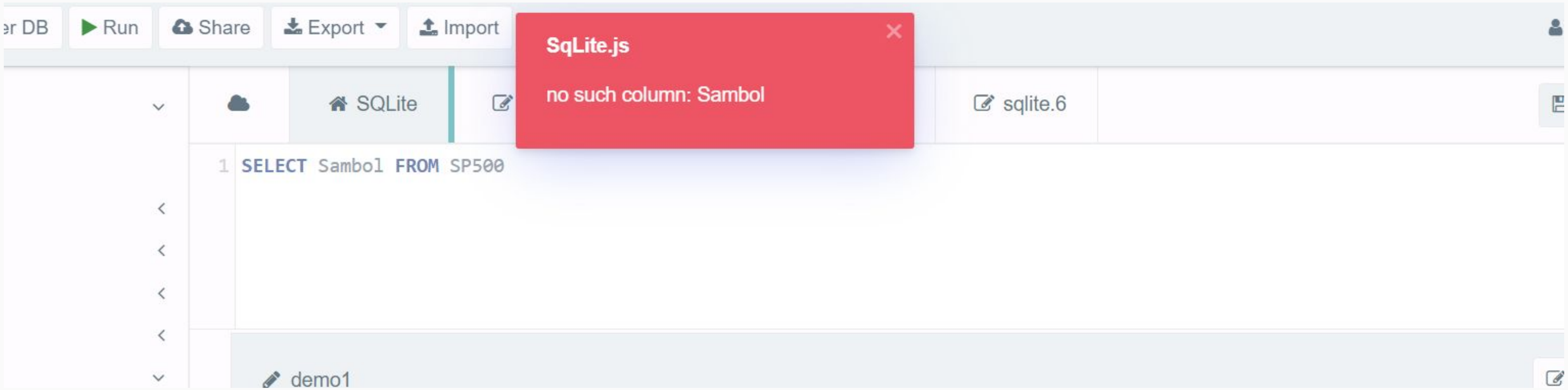
SQL FUNDAMENTALS: SELECT statement

SELECT …. FROM ….

- *SELECT * FROM table*
- *SELECT column name FROM table*
- *SELECT Column1, Column2, Column3 FROM table*
- Comments (-- , /* ….. */)
- Code formatting:
  - **Different columns should be separated by comma!**
  - The SQL Keywords are case-insensitive (SELECT , FROM , WHERE , etc), but are often written in all caps as a good practice
  - Usually the column and table name are also case-insensitive

SQL FUNDAMENTALS - Errors

- **Getting errors is absolutely OK**
- Usually IDE have suggestions of what is the type of error



SqLite.js

no such column: Sambol

```
SELECT Sambol FROM SP500
```

DEMO 1: SELECT statement

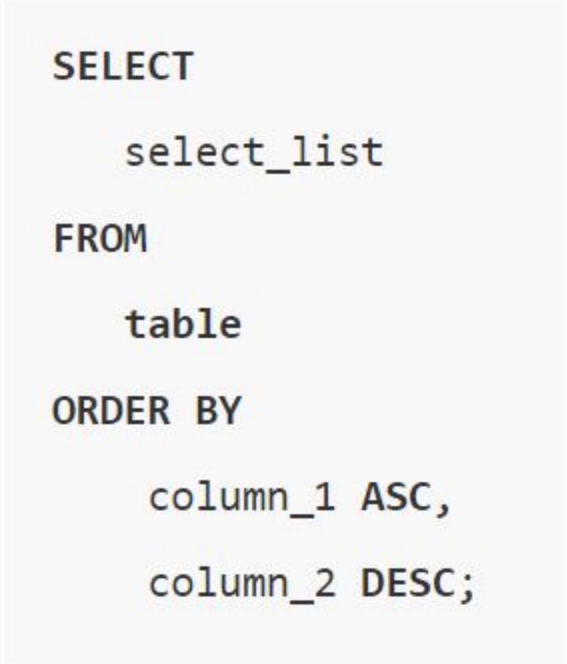
1. Please write a query that extracts all columns and rows from AccountManagers table.
2. Please extract columns CustomerID, CustomerEmail, AccountManagerPhone from Customers table. Please format the SQL query using the best practices.
3. Please select columns Customer, CountryResidence, AmountPosition from Customers table. What error do you get?

SQL Fundamentals

- SELECT statement
- **ORDER BY, LIMIT clauses**
- WHERE and Logical Operators

SQL FUNDAMENTALS: ORDER BY

- ORDER BY clause is sorting the result based on one or more columns in different order.
- Column name by which you want to sort after the ORDER BY clause followed by the ASC or DESC keyword.
    - The ASC keyword means ascending.
    - And the DESC keyword means descending.
- If not specified ASC order is by default
- You can sort the result set using a column that does not appear in the select list of the SELECT clause.
- Use a comma (,) to separate multiple order columns



```
SELECT
   select_list
FROM
   table
ORDER BY
    column_1 ASC,
    column_2 DESC;
```

SQL FUNDAMENTALS: LIMIT

- The LIMIT clause is an optional part of the SELECT statement. You use the LIMIT clause to constrain the number of rows returned by the query.
- We can retrieve 10 rows instead of 1 MLN rows
- The **row_count** is a positive integer that specifies the number of rows returned (5, 10, 20, 1000, etc.)

```
SELECT
        column_list
FROM
        table
LIMIT row_count;
```

DEMO 2: ORDER BY and LIMIT clauses

1. Please extract CustomerID, CountryResidence and CustomerAge sorting by CountryResidence in alphabetical order and CustomerAge from oldest to youngest using Customers table. What is the **highest age** of the customers in Canada?

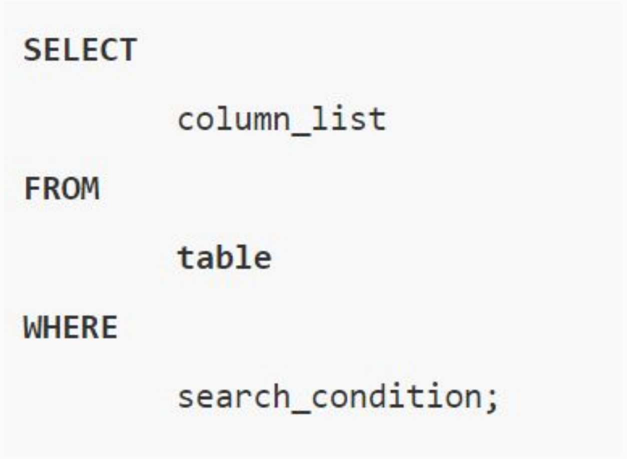
2. Please extract TOP 10 highest amounts using Amount Positions table. What is the **CUSTOMERID** on Place 10?

SQL Fundamentals

- SELECT statement
- ORDER BY, LIMIT clauses
- **WHERE and Logical Operators**

SQL FUNDAMENTALS: WHERE

- The WHERE clause is an optional clause of the SELECT statement. It appears after the FROM clause as the following statement
- When evaluating a SELECT statement with a WHERE clause, SQLite uses the following steps:
  - First, check the table in the FROM clause.
  - Second, evaluate the conditions in the WHERE clause to get the rows that met these conditions.
  - Third, make the final result set based on the rows in the previous step with columns in the SELECT clause.



```
SELECT
        column_list
FROM
        table
WHERE
        search_condition;
```

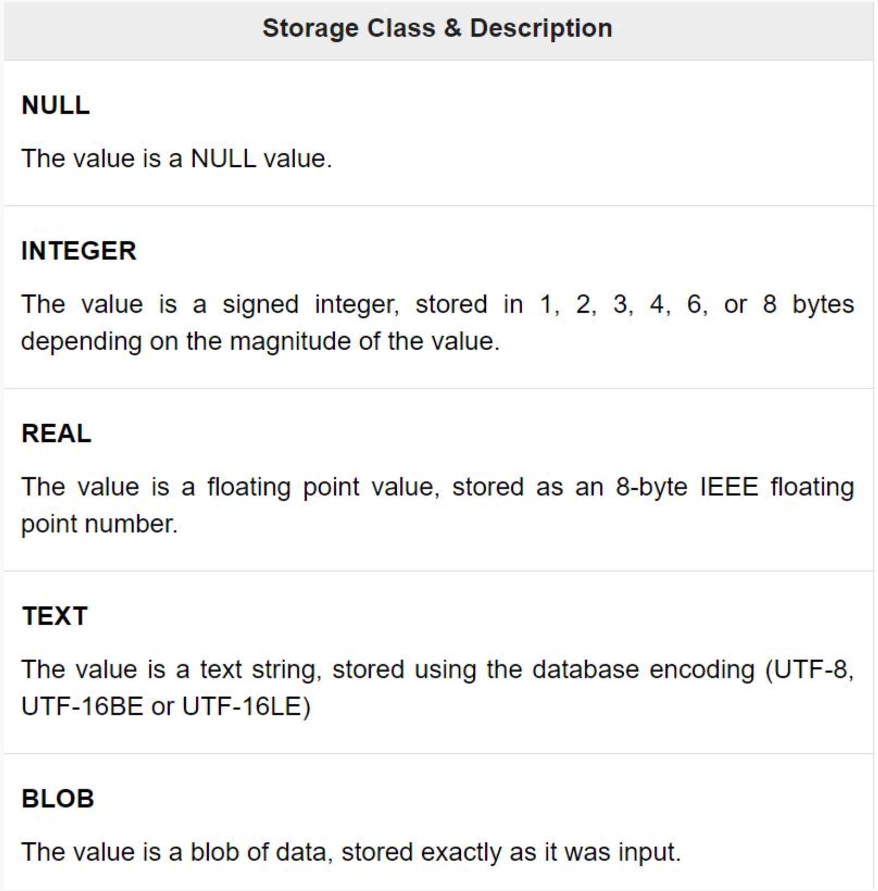
SQL FUNDAMENTALS: WHERE

For example, you can form a search condition as follows:

- WHERE column_1 = 100;
- WHERE column_2 **IN** (1,2,3); **IN** (‘Canada’, ‘UK’); **NOT IN** (‘Canada’, ‘Bulgaria’)
- WHERE column_3 **LIKE** 'An%';
- ‘%text%’ (will look for ‘text’ anywhere in the text); **NOT LIKE**
- WHERE column_4 **BETWEEN** 10 AND 20;
- WHERE column_5 **IS NULL**; IS **NOT NULL**
- WHERE column_6 <> “”

- Use **AND** or **OR** operator for multiple conditions

SQL FUNDAMENTALS: Data Types



Storage Class & Description
NULL The value is a NULL value.
INTEGER The value is a signed integer, stored in 1, 2, 3, 4, 6, or 8 bytes depending on the magnitude of the value.
REAL The value is a floating point value, stored as an 8-byte IEEE floating point number.
TEXT The value is a text string, stored using the database encoding (UTF-8, UTF-16BE or UTF-16LE)
BLOB The value is a blob of data, stored exactly as it was input.

IS NULL / IS NOT NULL

=, <, >, <>, etc.

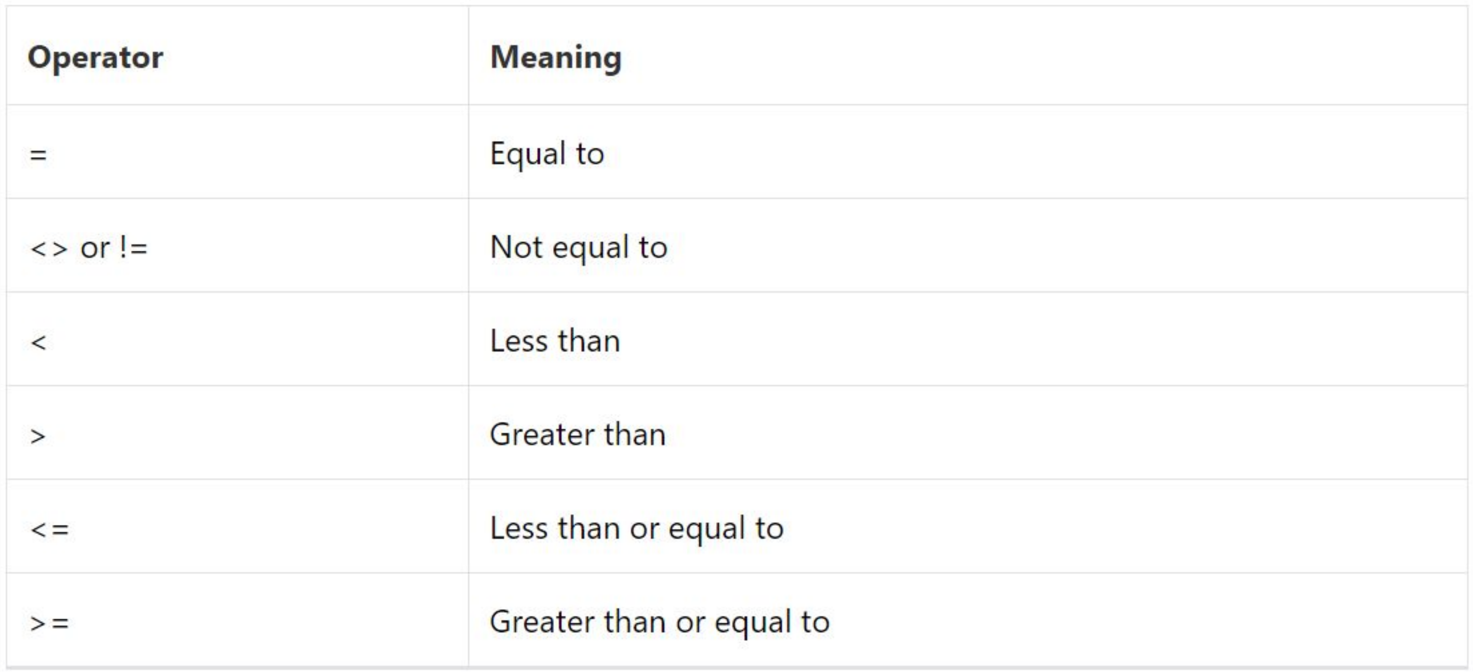
=, <, >, <>, etc.

=, <> (use ‘text’ or “text”)
IS CASE-SENSITIVE

Use “” for blanks

SQL FUNDAMENTALS: SQL Comparison Operators

- A comparison operator tests if two expressions are the same. The following table illustrates the comparison operators that you can use to construct expressions:



Operator	Meaning
=	Equal to
<> or !=	Not equal to
<	Less than
>	Greater than
<=	Less than or equal to
>=	Greater than or equal to

PRACTICE: WHERE and Logical Operators

1. Please select all columns from Customers table with Age more than 62. How many customers are above age 62?
2. Your manager asked you to find out WHAT AGE is the youngest customer who either lives in United Kingdom or his manager is Green.
3. How many customers both have Emails on .edu domain and and the Country of Residence is known? (Hint: you can use COUNT(*) to count the number of records)
4. How many customers have Stock positions between 2000 and 3000.

SQL CLAUSES Wrap UP

- SELECT
- ORDER BY
- LIMIT
- WHERE
- Comparison Operators
- IN
- LIKE
- BETWEEN
- IS NULL/NOT NULL
- DISTINCT
- CASE
- GROUP BY
- Aggregation Functions (MIN, MAX, SUM, AVG, COUNT)
- HAVING
- DATE()

SQL Intermediate

- DISTINCT
- CASE
- GROUP BY
- HAVING
- DATE

SQL Intermediate: DISTINCT

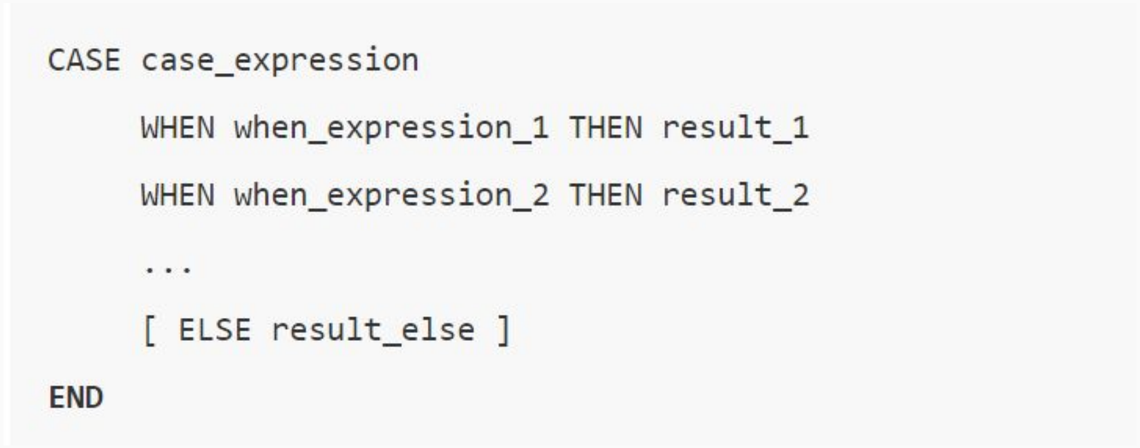
```
SELECT DISTINCT select_list

FROM table;
```

- The DISTINCT clause allows you to remove the duplicate rows in the result set.

    - SELECT DISTINCT (*)
    - SELECT DISTINCT(column1)
    - SELECT COUNT (DISTINCT column1)
    - SELECT COUNT (DISTINCT *)

SQL Intermediate: CASE

- CASE expression evaluates a list of conditions and returns an expression based on the result of the evaluation.
- he CASE expression is similar to the IF-THEN-ELSE statement in other programming languages.



```
CASE case_expression
     WHEN when_expression_1 THEN result_1
     WHEN when_expression_2 THEN result_2
     ...
     [ ELSE result_else ]
END
```

SQL Intermediate: CASE

- In case no case_expression matches the when_expression, the CASE expression returns the result_else in the ELSE clause. If you omit the ELSE clause, the CASE expression returns NULL.

DEMO 4 DISTINCT , CASE

1. Please extract how many **unique** Portfolio Positions have funds assigned in the database, compare it to the total possible Portfolio Positions in Portfolio Positions table. How many portfolio positions are not “active”?
2. How many customers have at least 1 active Portfolio Position?
3. How many customers have funds on Portfolio Position: Cash Remaining (hint: check the ID of this position in Portfolio Positions table)?
4. Please extract column Account Manager and produce new column where in case Account Manager is Mellow, type Mellow, and if someone else type OTHER. Rename the new column as ‘Managers’. Sort the output by Age ascending order. What is the first result in this new column?

SQL Intermediate: GROUP BY

- The GROUP BY clause a selected group of rows into summary rows by values of one or more columns.
- For each group, you can apply an aggregate function such as MIN, MAX, SUM, COUNT, or AVG to provide more information about each group.

```
SELECT
    column_1,
    aggregate_function(column_2)
FROM
    table
GROUP BY
    column_1,
    column_2;
```

SQL Intermediate: GROUP BY

- The GROUP BY clause comes after the FROM clause of the SELECT statement. In case a statement contains a WHERE clause, the GROUP BY clause must come after the WHERE clause.
- Following the GROUP BY clause is a column or a list of comma-separated columns used to specify the group.

DEMO 5 GROUP BY

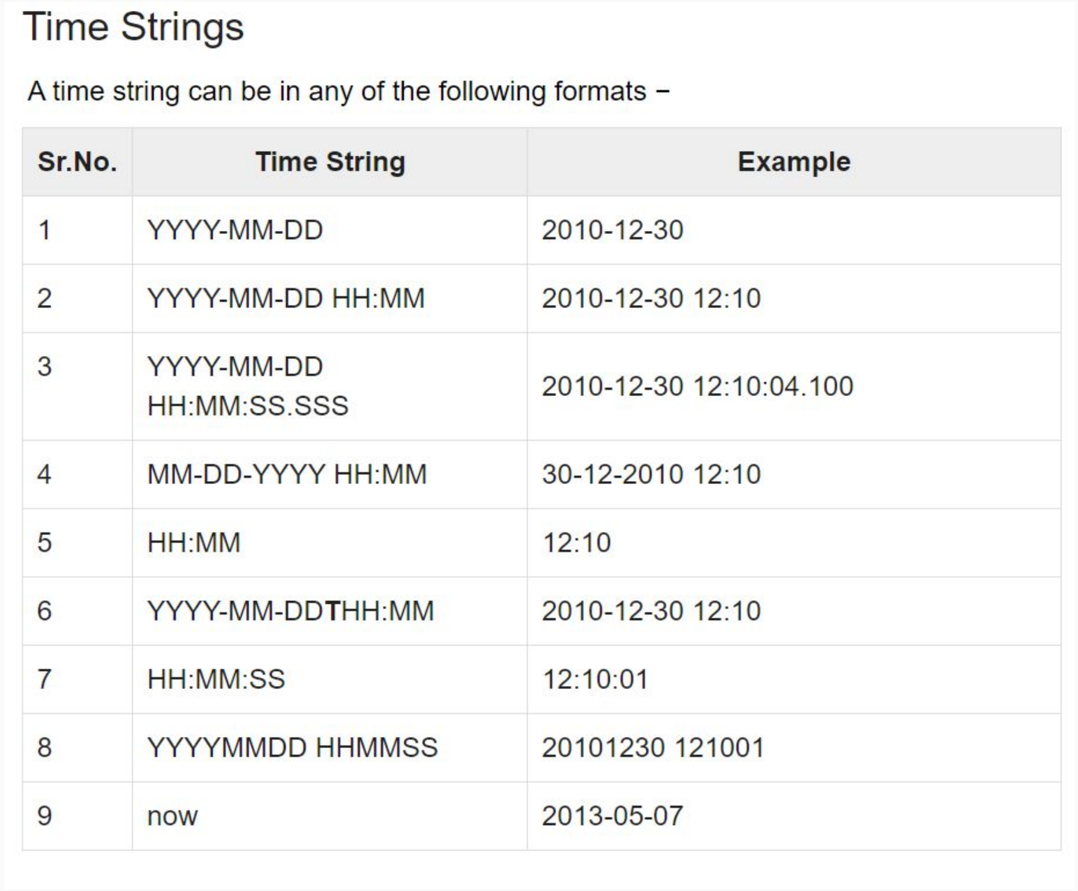
Your manager would like to analyze the **Customer Classes (Bottom, Middle and TOP)** and asked you to produce a report which can answer next questions:

- How many customers are inside each class?
- What is the average age of customers of each class?
- What is the maximum age of customers of each class?
- What is the minimum age of customers of each class?
- How many different countries do customers represent in each class?
- What is the number of managers having clients in each class?
- Please give each column a good looking name for the report purposes.

SQL Intermediate

- DISTINCT
- CASE
- GROUP BY
- DATE
- HAVING

SQL Intermediate: DATE, DATETIME



Time Strings

A time string can be in any of the following formats –

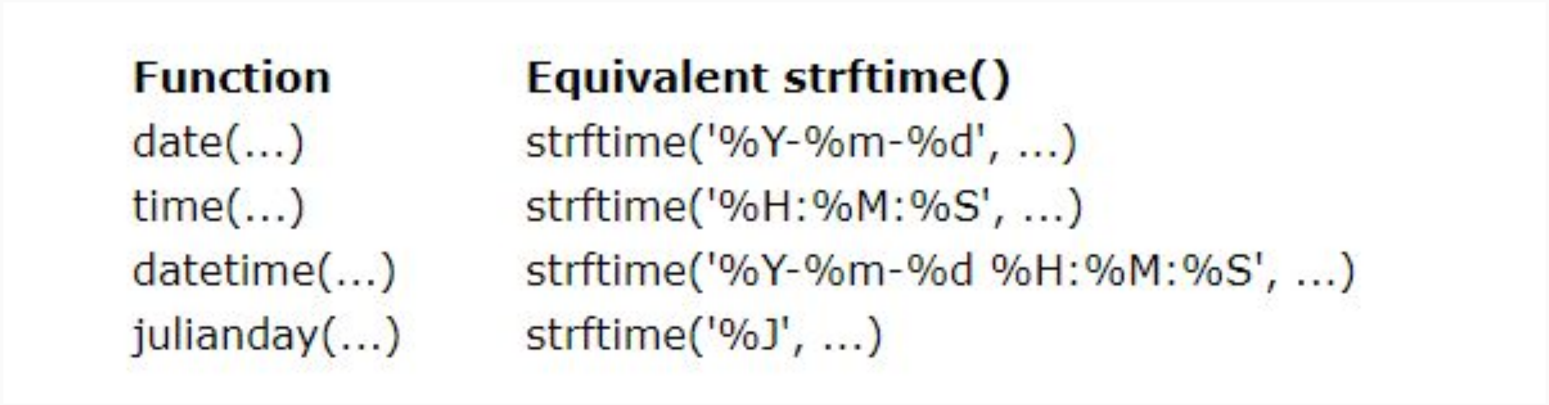
Sr.No.	Time String	Example
1	YYYY-MM-DD	2010-12-30
2	YYYY-MM-DD HH:MM	2010-12-30 12:10
3	YYYY-MM-DD HH:MM:SS.SSS	2010-12-30 12:10:04.100
4	MM-DD-YYYY HH:MM	30-12-2010 12:10
5	HH:MM	12:10
6	YYYY-MM-DD**T**HH:MM	2010-12-30 12:10
7	HH:MM:SS	12:10:01
8	YYYYMMDD HHMMSS	20101230 121001
9	now	2013-05-07

SQL Intermediate: DATE, DATETIME

- The date and time functions use a subset of IS0-8601 date and time formats.
- The date() function returns the date in this format: YYYY-MM-DD. The time() function returns the time as HH:MM:SS.
- The datetime() function returns "YYYY-MM-DD HH:MM:SS".
- The julianday() function returns the Julian day - the number of days since noon in Greenwich on November 24, 4714 B.C. (Proleptic Gregorian calendar).
- **The strftime()** routine returns the date formatted according to the format string specified as the first argument

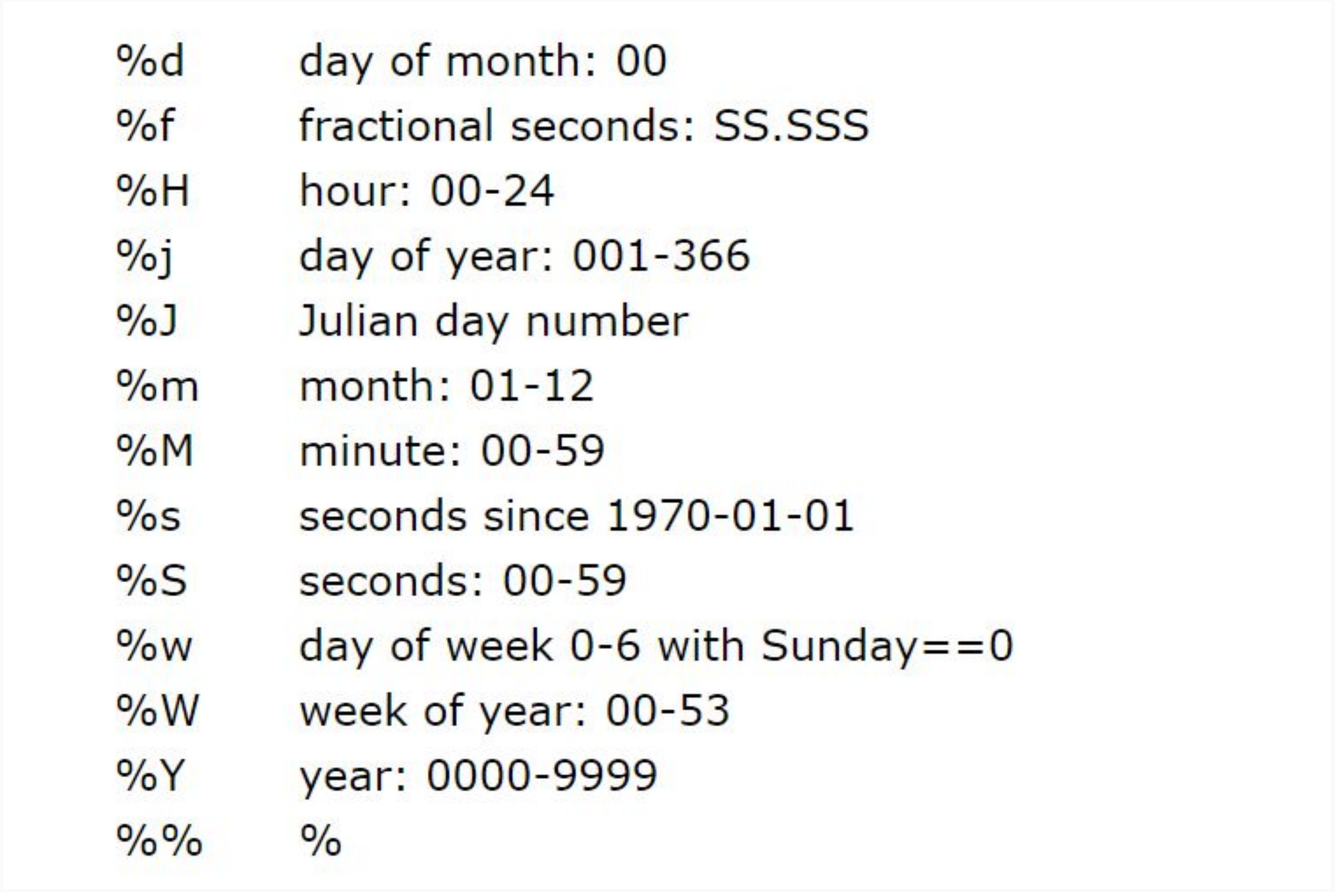
SQL Intermediate: DATE, DATETIME

- SQLite does not support built-in date and/or time storage class. Instead, it leverages some built-in date and time functions to use other storage classes such as TEXT, REAL, or INTEGER for storing the date and time values.



Function	Equivalent strftime()
date(...)	strftime('%Y-%m-%d', ...)
time(...)	strftime('%H:%M:%S', ...)
datetime(...)	strftime('%Y-%m-%d %H:%M:%S', ...)
julianday(...)	strftime('%J', ...)

SQL Intermediate: DATE, DATETIME



%d	day of month: 00
%f	fractional seconds: SS.SSS
%H	hour: 00-24
%j	day of year: 001-366
%J	Julian day number
%m	month: 01-12
%M	minute: 00-59
%s	seconds since 1970-01-01
%S	seconds: 00-59
%w	day of week 0-6 with Sunday==0
%W	week of year: 00-53
%Y	year: 0000-9999
%%	%

PRACTICE: Task 6 DATES

- 1. Please find what is the first date of registration of any customer?
- 2. Please calculate how many days passed since the newest customer was registered?
- 3. How many customers registered in 2019 are from UK? (WHERE clause)

SQL Intermediate: HAVING

- The HAVING clause specifies a search condition for a group.
- You often use the HAVING clause with the GROUP BY clause. The GROUP BY clause groups a set of rows into a set of summary rows or groups. Then the HAVING clause filters groups based on a specified condition.
- If you use the HAVING clause, you must include the GROUP BY clause; otherwise, you will get the following error:

  ```
  Error: a GROUP BY clause is required before HAVING
  ```
- **Note:** that the HAVING clause is applied after GROUP BY clause, whereas the WHERE clause is applied before the GROUP BY clause.

SQL Intermediate: SUBQUERY - chain type

- Subquery - chain type: SELECT statement inside another SELECT statement

```
SELECT column1, column 2
FROM (
        SELECT columnA, columnB
        FROM table )
```

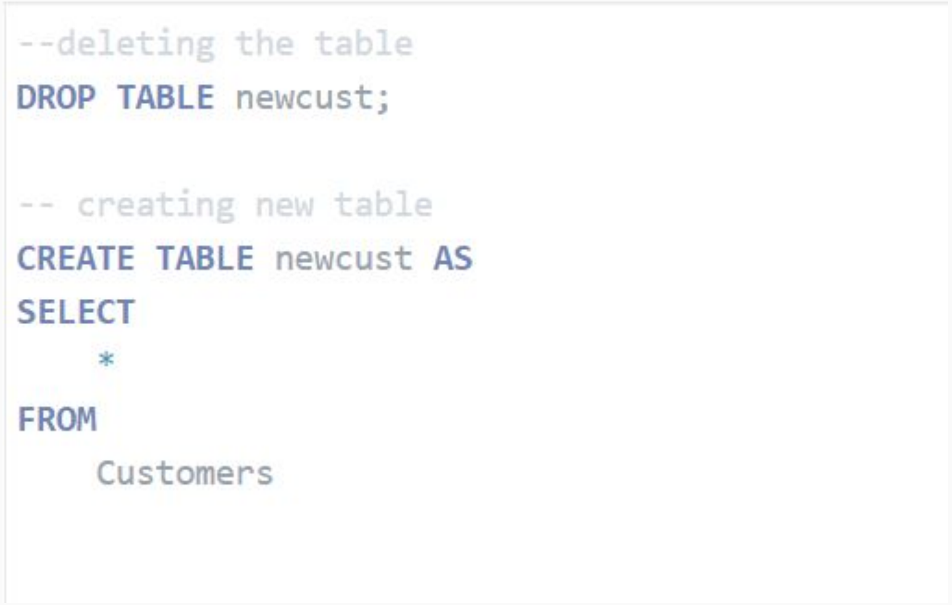
SQL Intermediate: SUBQUERY - WITH

- Subquery - WITH: creates a table in the query memory

```
WITH tablename AS
(SELECT column1
FROM table)


SELECT column1
FROM tablename
```

SQL Intermediate: CREATE

- CREATE clause is used to create and record new table
- If the table already exists it should be deleted using DROP first before it can be created again with a same name.



```
--deleting the table
DROP TABLE newcust;

-- creating new table
CREATE TABLE newcust AS
SELECT
    *
FROM
    Customers
```

DEMO 7 HAVING and Subquery WITH

- Please compose a report of total amount funded by customer (use CustomerID here) only with customers who have positions for more than 70000 in total.
- Create 2 queries: with subquery WITH and HAVING (use ; to split queries)

SQL Advanced

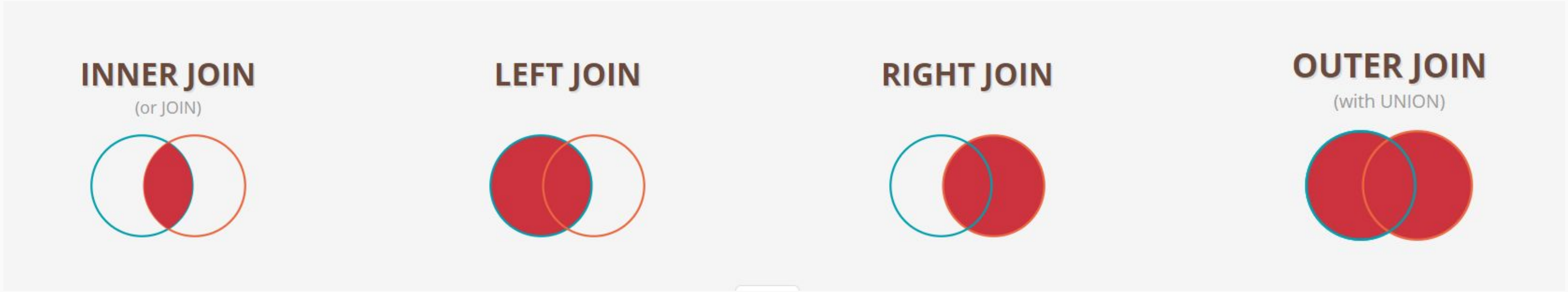
- INNER JOIN
- LEFT JOIN
- RIGHT JOIN
- FULL OUTER JOIN
- CROSS JOIN
- UNION

Objective

Be able to provide answer to meaningful questions by connecting the information between separate tables using JOIN clauses:

- Understand difference between different JOIN queries;
- Learn the JOIN query syntax;
- Be able to select which JOIN query is appropriate;
- Learn to combine information from different tables;
- Be able to combine JOIN, condition and aggregation functions in one query

SQL Advanced: Types of JOINS



CROSS JOIN

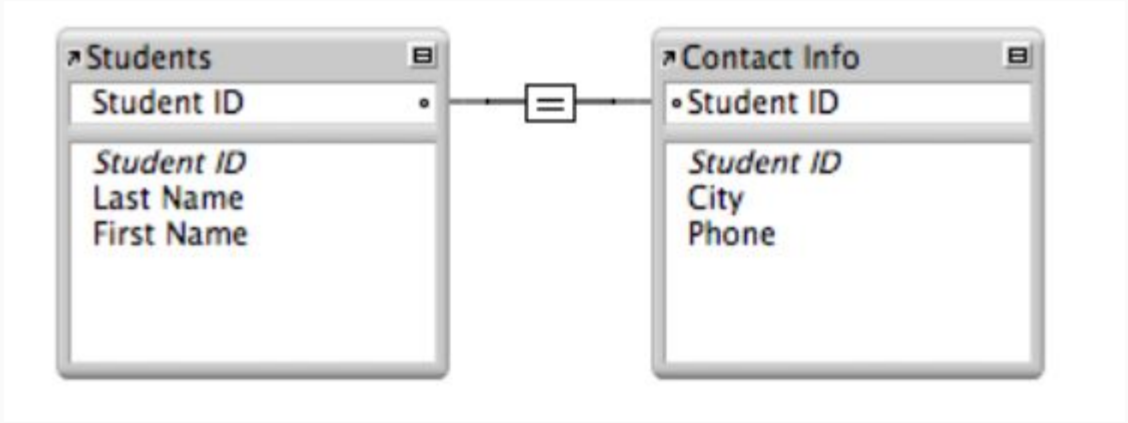
*SQLite doesn’t directly support the RIGHT JOIN and FULL OUTER JOIN

SQL Advanced: Types of Relationship

- **One-to-one relationship**
- **One-to-many relationship** (or many-to-one)
- **Many-to-many relationship**

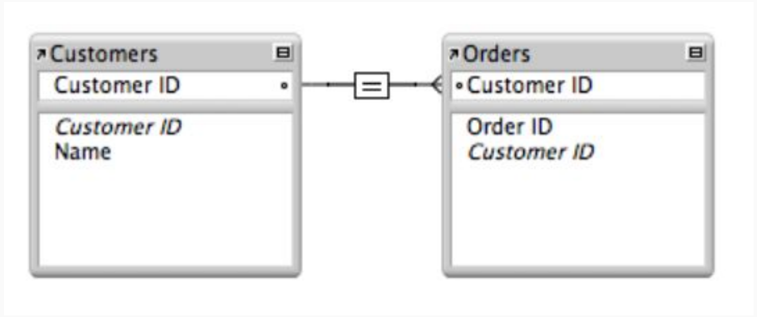
SQL Advanced: Types of Relationship: One-to-one

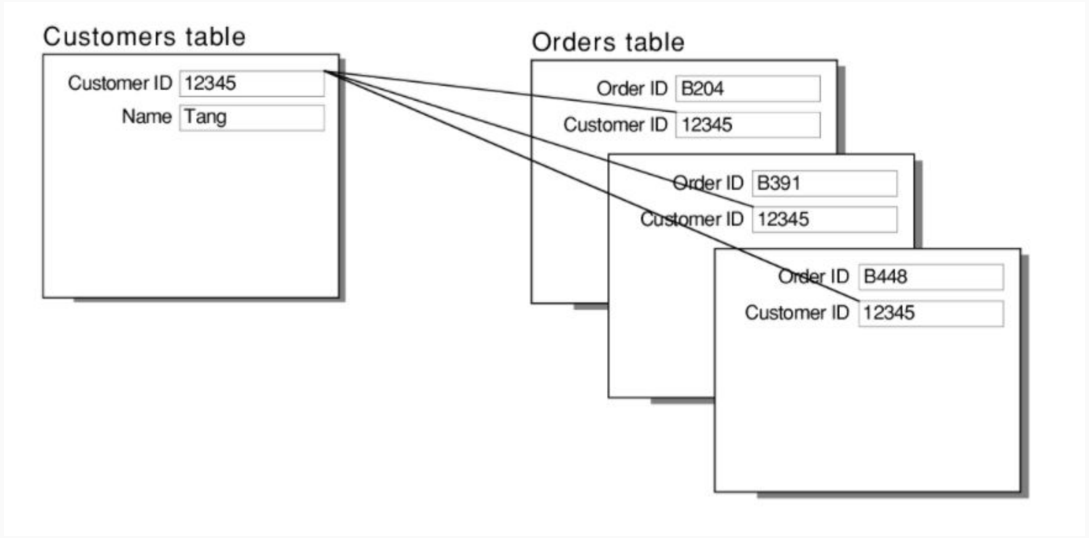
- In a **one-to-one relationship**, one record in a table is associated with one and only one record in another table.
- For example, in a school database, each student has only one student ID, and each student ID is assigned to only one person.



SQL Advanced: Types of Relationship: One-to-many

- In a **one-to-many relationship**, one record in a table can be associated with one or more records in another table.
- But one record from a second table is associated to a single record from the first table
- For example, each customer can have many sales orders.





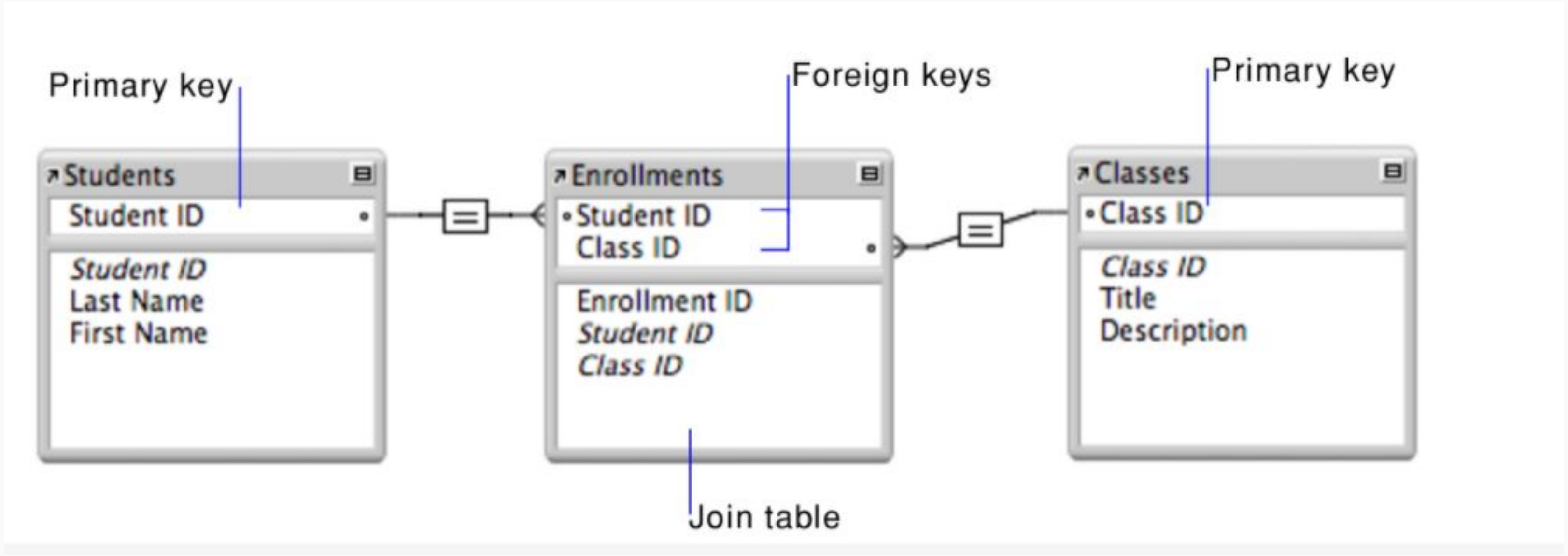
SQL Advanced: Types of Relationship: Many-to-many

- A **many-to-many relationship** occurs when multiple records in a table are associated with multiple records in another table.
- For example, a many-to-many relationship exists between customers and products: customers can purchase various products, and products can be purchased by many customers.

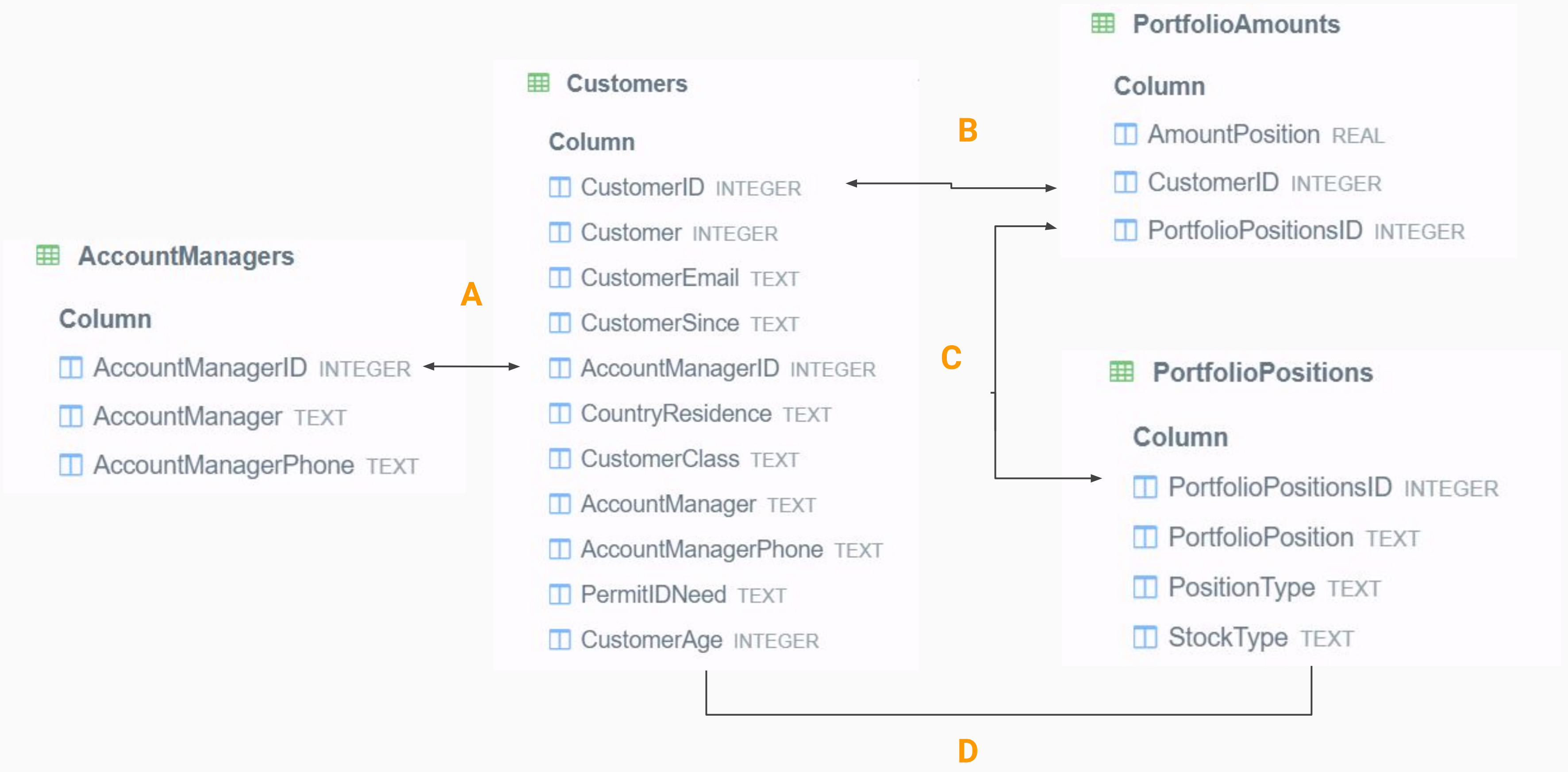
- In relational databases many-to-many relationship is usually represented by two one-to-many relationships by using a third table.
- Each record in a join table includes a match field that contains the value of the ***primary keys*** of the two tables it joins. (In the join table, these match fields are ***foreign keys***.) These foreign key fields are populated with data as records in the join table are created from either table it joins.

SQL Advanced: Types of Relationship: Many-to-many

- Another typical example - students and classes: a student can register for many classes, and a class can include many students.

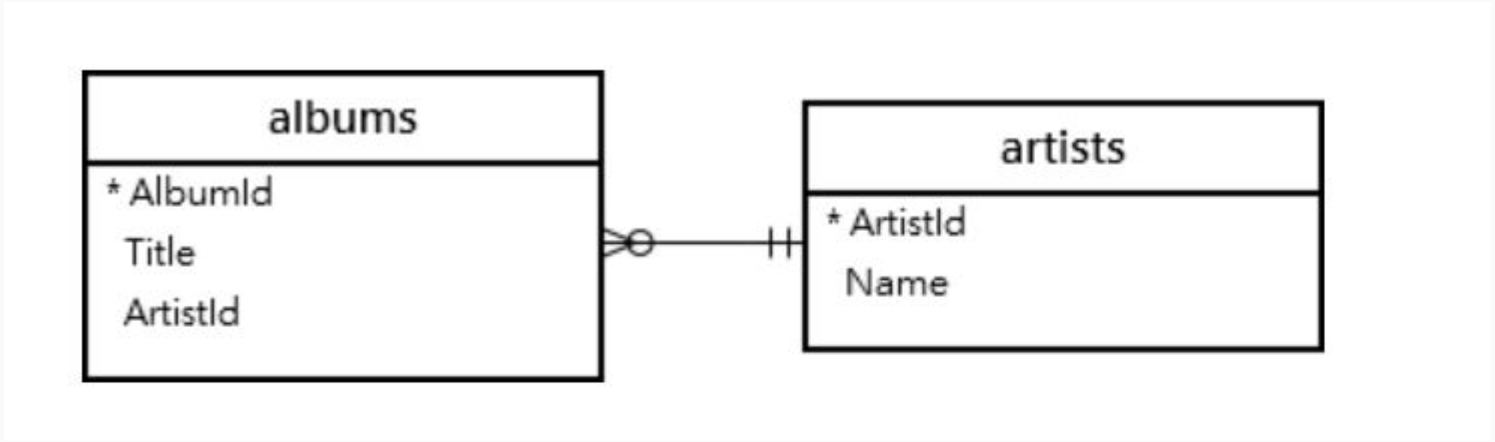


SQL Advanced: BROKERAGE DATASET - Types of Relationships

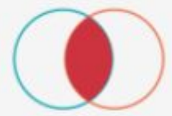


SQL Advanced: Demo dataset

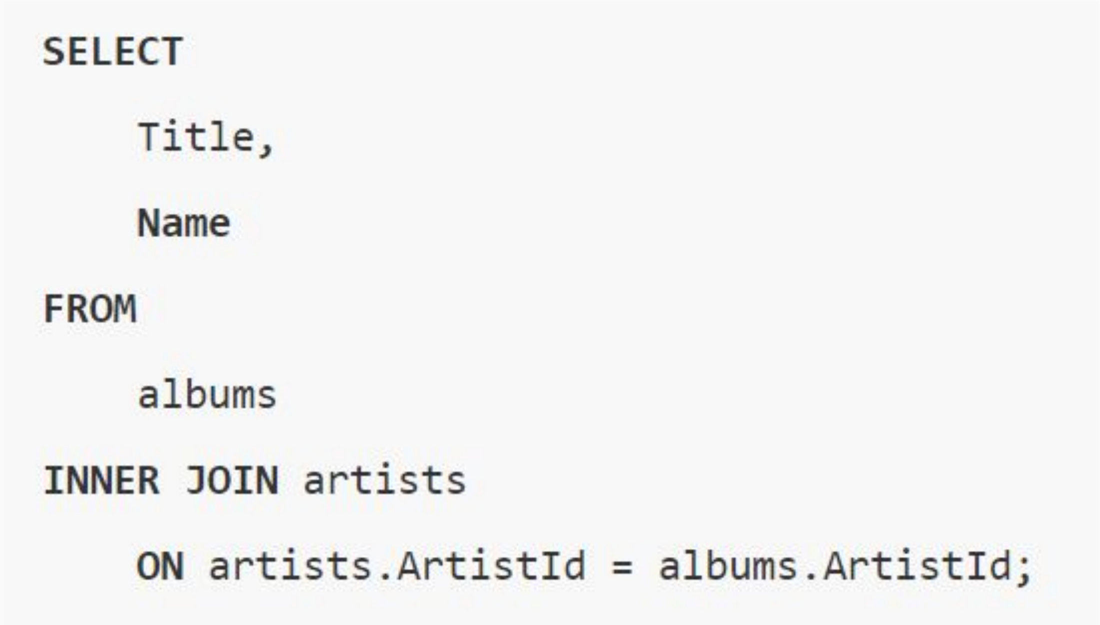
- Artists and albums tables: an artist can have zero or many albums while an album belongs to one artist.



SQL Advanced: INNER JOIN (JOIN)



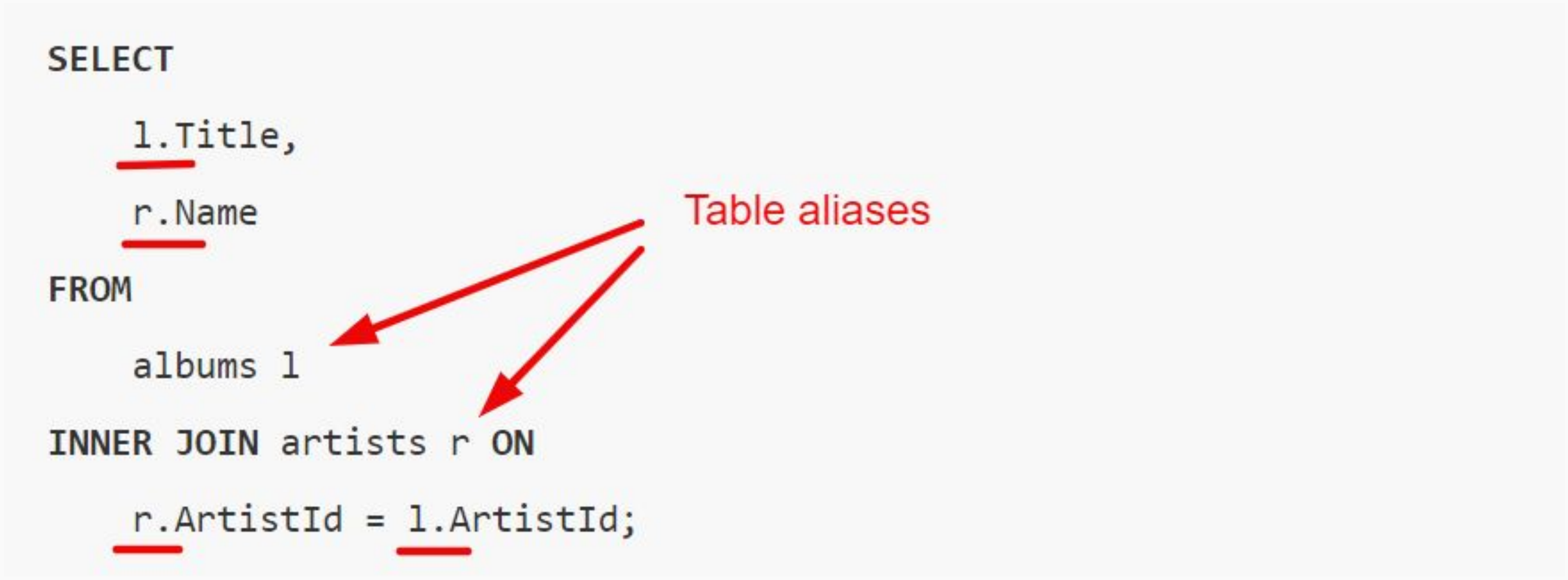
- INNER JOIN or JOIN returns records that match in **BOTH** right and left tables.
- INNER JOIN clause matches each row from the albums table with every row from the artists table based on the join condition (artists.ArtistId = albums.ArtistId) specified after the ON keyword.



```
SELECT
    Title,
    Name
FROM
    albums
INNER JOIN artists
    ON artists.ArtistId = albums.ArtistId;
```

SQL Advanced: INNER JOIN (JOIN) - aliases

- This query uses table **aliases** (l for the albums table and r for artists table) to shorten the query:



```
SELECT
    l.Title,
    r.Name
FROM
    albums l
INNER JOIN artists r ON
    r.ArtistId = l.ArtistId;
```

SQL Advanced: INNER JOIN (JOIN) - USING syntax

- In case the column names of joined tables are the same e.g., ArtistId, you can use the **USING** syntax
- The clause USING(ArtistId) is equivalent to the clause ON artists.ArtistId = albums.ArtistId

```
SELECT
    Title,
    Name
FROM
    albums
INNER JOIN artists USING(ArtistId);
```

Contacts:

Sergey Vichev

Ph.D candidate

www.linkedin.com/in/sergeyvichev/

THANK YOU