In [1]:
```
%reload_ext sql
%config SqlMagic.autopandas = True
%config SqlMagic.feedback = False
%config SqlMagic.style = 'default'  # This is the key fix
```

In [2]:
```
# # First, close all SQL connections
# %sql --close sqlite:///practice3.db
```

In [3]:
```
# # Delete the database file
# !rm -f practice3.db
```

In [4]:
```
# Connect to an SQLite database (creates if doesn't exist)
%sql sqlite:///practice3.db
```

SQL Injection

SQL injection is a code injection technique that might destroy your database.

SQL injection is one of the most common web hacking techniques.

SQL injection is the placement of malicious code in SQL statements, via web page input.

SQL injection usually occurs when you ask a user for input, like their username/userid, and instead of a name/id, the user gives you an SQL statement that you will unknowingly run on your database.

Look at the following example which creates a SELECT statement by adding a variable (txtUserId) to a select string.

The variable is fetched from user input (getRequestString):

txtUserId = getRequestString("UserId"); txtSQL = "SELECT * FROM Users WHERE UserId = " + txtUserId;

The original purpose of the code was to create an SQL statement to select a user, with a given user id.

If there is nothing to prevent a user from entering "wrong" input, the user can enter some "smart" input like this:

HTML Inputbox : UserId: 105 OR 1=1

Then, the SQL statement will look like this:

SELECT * FROM Users WHERE UserId = 105 OR 1=1;

The SQL above is valid and will return ALL rows from the "Users" table, since OR 1=1 is always TRUE.

Does the example above look dangerous? What if the "Users" table contains names and passwords?

The SQL statement above is much the same as this:

SELECT UserId, Name, Password FROM Users WHERE UserId = 105 or 1=1;

Use SQL Parameters for Protection:

To protect a web site from SQL injection, you can use SQL parameters.

SQL parameters are values that are added to an SQL query at execution time, in a controlled manner.

ASP.NET Razor Example txtUserId = getRequestString("UserId"); txtSQL = "SELECT * FROM Users WHERE UserId = @0"; db.Execute(txtSQL,txtUserId);

Advanced SQL

In SQL, a window function or analytic function is a function which uses values from one or multiple rows to return a value for each row. (This contrasts with an aggregate function, which returns a single value for multiple rows.)

Window functions have an OVER clause; any function without an OVER clause is not a window function, but rather an aggregate or single-row (scalar) function.

Rank | Dense Rank | Row Number

In [5]:
```
%%sql
create table if not exists Grade (Marks integer);
```

```
* sqlite:///practice3.db
```

Out[5]: —

In [6]:
```
%sql select * from Grade;
```

```
* sqlite:///practice3.db
```

Out[6]:

	Marks
0	85
1	78
2	68
3	68
4	65
5	85

In [7]:
```
%sql SELECT name FROM sqlite_master WHERE type='table';
```

```
* sqlite:///practice3.db
```

Out[7]:

	name
0	Grade
1	Dept
2	DeptSales
3	StateSales
4	MovingAvg
5	Stocks
6	NewStocks
7	TrainJourney
8	NewTrainJourney
9	Players
10	marks
11	JoinsDemo1
12	JoinsDemo2
13	JoinsDemo3
14	JoinsDemo4
15	TimeDiffDemo
16	ListAggDemo
17	Customers
18	Orders
19	employees
20	users1
21	users
22	transactions
23	sales
24	orders1

In [8]:
```
%sql insert into Grade (Marks) values (78), (68), (68), (65), (85);
```

```
* sqlite:///practice3.db
```

Out[8]: —

In [9]:
```
%sql select * from Grade;
```

```
* sqlite:///practice3.db
```

Out[9]:

	Marks
0	85
1	78
2	68
3	68
4	65
5	85
6	78
7	68
8	68
9	65
10	85

In [10]:
```
%%sql
select *,
rank() over(order by Marks desc) rnk,
dense_rank() over(order by Marks desc) dense_rnk,
row_number() over(order by Marks desc) row_number

from Grade;
```

```
* sqlite:///practice3.db
```

Out[10]:

	Marks	rnk	dense_rnk	row_number
0	85	1	1	1
1	85	1	1	2
2	85	1	1	3
3	78	4	2	4
4	78	4	2	5
5	68	6	3	6
6	68	6	3	7
7	68	6	3	8
8	68	6	3	9
9	65	10	4	10
10	65	10	4	11

In [11]:
```
%%sql
create table if not exists Dept (ID integer primary key, Department text,
```

```
* sqlite:///practice3.db
```

Out[11]: —

In [12]:
```
%%sql
insert into Dept values
(1, 'Sales', 1000),
(2,'IT',1500),
(3,'Sales',2000),
(4,'Sales',1700),
(5, 'IT',1800),
(6,'Accounts',1200),
(7,'Accounts',1100);
```

```
 * sqlite:///practice3.db
(sqlite3.IntegrityError) UNIQUE constraint failed: Dept.ID
[SQL: insert into Dept values
(1, 'Sales', 1000),
(2,'IT',1500),
(3,'Sales',2000),
(4,'Sales',1700),
(5, 'IT',1800),
(6,'Accounts',1200),
(7,'Accounts',1100);]
(Background on this error at: https://sqlalche.me/e/20/gkpj)
```

In [13]:
```
%sql select * from Dept;
```

```
 * sqlite:///practice3.db
```

Out[13]:

	ID	Department	Salary
0	1	Sales	1000
1	2	IT	1500
2	3	Sales	2000
3	4	Sales	1700
4	5	IT	1800
5	6	Accounts	1200
6	7	Accounts	1100

In [14]:
```
%%sql
select *, rank() over (partition by Department order by Salary desc)
as emp_rank from Dept;
```

```
 * sqlite:///practice3.db
```

Out[14]:

	ID	Department	Salary	emp_rank
0	6	Accounts	1200	1
1	7	Accounts	1100	2
2	5	IT	1800	1
3	2	IT	1500	2
4	3	Sales	2000	1
5	4	Sales	1700	2
6	1	Sales	1000	3

Rows Between clause in SQL

In [15]:
```
%%sql
create table if not exists DeptSales (ID integer primary key,
Date date, Sales integer);
```

```
 * sqlite:///practice3.db
```

Out[15]: —

In [16]:
```
%%sql
insert into DeptSales values
(1, '22-06-2022',603),
(2,'21-06-2022',478),
(3,'20-06-2022',679),
(4,'19-06-2022',443),
(5,'18-06-2022',540),
(6,'17-06-2022',740),
(7,'16-06-2022',850),
(8,'15-06-2022',604),
(9,'14-06-2022',339),
(10,'13-06-2022',905);
```

```
 * sqlite:///practice3.db
(sqlite3.IntegrityError) UNIQUE constraint failed: DeptSales.ID
[SQL: insert into DeptSales values
(1, '22-06-2022',603),
(2,'21-06-2022',478),
(3,'20-06-2022',679),
(4,'19-06-2022',443),
(5,'18-06-2022',540),
(6,'17-06-2022',740),
(7,'16-06-2022',850),
(8,'15-06-2022',604),
(9,'14-06-2022',339),
(10,'13-06-2022',905);]
(Background on this error at: https://sqlalche.me/e/20/gkpj)
```

In [17]:
```
%sql select * from DeptSales;
```

```
 * sqlite:///practice3.db
```

Out[17]:

	ID	Date	Sales
0	1	22-06-2022	603
1	2	21-06-2022	478
2	3	20-06-2022	679
3	4	19-06-2022	443
4	5	18-06-2022	540
5	6	17-06-2022	740
6	7	16-06-2022	850
7	8	15-06-2022	604
8	9	14-06-2022	339
9	10	13-06-2022	905

Data we want: current sales + prev. day sales+ next day sales for each row except first and last row

So, in general, we want current row value+ m rows values preceding + n rows values following.

In [18]:
```
%%sql
select *,Sum(Sales) over (order by Date rows between 1 preceding and
1 following) as total_sales_today_yesterday_nextday from DeptSales;
```

```
* sqlite:///practice3.db
```

Out[18]:

	ID	Date	Sales	total_sales_today_yesterday_nextday
0	10	13-06-2022	905	1244
1	9	14-06-2022	339	1848
2	8	15-06-2022	604	1793
3	7	16-06-2022	850	2194
4	6	17-06-2022	740	2130
5	5	18-06-2022	540	1723
6	4	19-06-2022	443	1662
7	3	20-06-2022	679	1600
8	2	21-06-2022	478	1760
9	1	22-06-2022	603	1081

SQL Query for sum of all rows before a particular row and the all rows after a particular row.

In [19]:
```
%%sql
select *,Sum(Sales) over (order by Date rows between unbounded
```

```
preceding and unbounded following) as total_sales_before_after
from DeptSales;
```

```
* sqlite:///practice3.db
```

Out[19]:

	ID	Date	Sales	total_sales_before_after
0	10	13-06-2022	905	6181
1	9	14-06-2022	339	6181
2	8	15-06-2022	604	6181
3	7	16-06-2022	850	6181
4	6	17-06-2022	740	6181
5	5	18-06-2022	540	6181
6	4	19-06-2022	443	6181
7	3	20-06-2022	679	6181
8	2	21-06-2022	478	6181
9	1	22-06-2022	603	6181

Cumulative Sum (Running Sum) in SQL

In [20]:
```
%%sql
select *,sum(Sales) over (order by Date rows between unbounded
preceding and current row) as running_sum from DeptSales;
```

```
* sqlite:///practice3.db
```

Out[20]:

	ID	Date	Sales	running_sum
0	10	13-06-2022	905	905
1	9	14-06-2022	339	1244
2	8	15-06-2022	604	1848
3	7	16-06-2022	850	2698
4	6	17-06-2022	740	3438
5	5	18-06-2022	540	3978
6	4	19-06-2022	443	4421
7	3	20-06-2022	679	5100
8	2	21-06-2022	478	5578
9	1	22-06-2022	603	6181

In [21]:
```
%%sql
create table if not exists StateSales(ID integer primary key,
State text, Date date, Sales);
```

```
* sqlite:///practice3.db
```

Out[21]: —

In [22]:
```
%%sql
insert into StateSales values
(1, 'Jharkhand', '22-06-2022',603),
(2, 'Jharkhand', '12-06-2022',683),
(3, 'Jharjhand', '21-06-2022',693),
(4, 'Bihar', '22-06-2022',603),
(5, 'Bihar', '22-06-2022',603),
(6, 'Bihar', '22-06-2022',603),
(7, 'Maharastra', '22-06-2022',603),
(8, 'Maharastra', '22-06-2022',603),
(9, 'Maharastra', '22-06-2022',603)
```

```
 * sqlite:///practice3.db
(sqlite3.IntegrityError) UNIQUE constraint failed: StateSales.ID
[SQL: insert into StateSales values
(1, 'Jharkhand', '22-06-2022',603),
(2, 'Jharkhand', '12-06-2022',683),
(3, 'Jharjhand', '21-06-2022',693),
(4, 'Bihar', '22-06-2022',603),
(5, 'Bihar', '22-06-2022',603),
(6, 'Bihar', '22-06-2022',603),
(7, 'Maharastra', '22-06-2022',603),
(8, 'Maharastra', '22-06-2022',603),
(9, 'Maharastra', '22-06-2022',603)]
(Background on this error at: https://sqlalche.me/e/20/gkpj)
```

In [23]:
```
%sql select * from StateSales
```

```
 * sqlite:///practice3.db
```

Out[23]:

	ID	State	Date	Sales
0	1	Jharkhand	22-06-2022	603
1	2	Jharkhand	12-06-2022	683
2	3	Jharjhand	21-06-2022	693
3	4	Bihar	22-06-2022	603
4	5	Bihar	22-06-2022	603
5	6	Bihar	22-06-2022	603
6	7	Maharastra	22-06-2022	603
7	8	Maharastra	22-06-2022	603
8	9	Maharastra	22-06-2022	603

In [24]:
```
%%sql
select *,sum(Sales) over (partition by State order by Date rows
between unbounded preceding and current row) as running_total_statewise
from StateSales;
```

```
 * sqlite:///practice3.db
```

Out[24]:

	ID	State	Date	Sales	running_total_statewise
0	4	Bihar	22-06-2022	603	603
1	5	Bihar	22-06-2022	603	1206
2	6	Bihar	22-06-2022	603	1809
3	3	Jharjhand	21-06-2022	693	693
4	2	Jharkhand	12-06-2022	683	683
5	1	Jharkhand	22-06-2022	603	1286
6	7	Maharastra	22-06-2022	603	603
7	8	Maharastra	22-06-2022	603	1206
8	9	Maharastra	22-06-2022	603	1809

First Value, Last Value and Nth Value in SQL

In [25]:
```
%%sql
select *, first_value(Sales) over (partition by State order by Date) firs
last_value(Sales) over (partition by State order by Date rows between unb
from StateSales
```

```
* sqlite:///practice3.db
```

Out[25]:

	ID	State	Date	Sales	first_day_sales	last_day_sales
0	4	Bihar	22-06-2022	603	603	603
1	5	Bihar	22-06-2022	603	603	603
2	6	Bihar	22-06-2022	603	603	603
3	3	Jharjhand	21-06-2022	693	693	693
4	2	Jharkhand	12-06-2022	683	683	603
5	1	Jharkhand	22-06-2022	603	683	603
6	7	Maharastra	22-06-2022	603	603	603
7	8	Maharastra	22-06-2022	603	603	603
8	9	Maharastra	22-06-2022	603	603	603

In [26]:
```
%%sql
select *, first_value(Sales) over (partition by State order by Date) firs
last_value(Sales) over (partition by State order by Date) last_day_sales
from StateSales
```

```
* sqlite:///practice3.db
```

Out[26]:

	ID	State	Date	Sales	first_day_sales	last_day_sales
0	4	Bihar	22-06-2022	603	603	603
1	5	Bihar	22-06-2022	603	603	603
2	6	Bihar	22-06-2022	603	603	603
3	3	Jharjhand	21-06-2022	693	693	693
4	2	Jharkhand	12-06-2022	683	683	683
5	1	Jharkhand	22-06-2022	603	683	603
6	7	Maharastra	22-06-2022	603	603	603
7	8	Maharastra	22-06-2022	603	603	603
8	9	Maharastra	22-06-2022	603	603	603

In [27]:
```
%%sql
SELECT *,
       nth_value(Sales, 2) OVER (
           PARTITION BY State
           ORDER BY Date
           ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING
       ) AS first_day_sales
FROM StateSales;
```

```
* sqlite:///practice3.db
```

Out[27]:

	ID	State	Date	Sales	first_day_sales
0	4	Bihar	22-06-2022	603	603.0
1	5	Bihar	22-06-2022	603	603.0
2	6	Bihar	22-06-2022	603	603.0
3	3	Jharjhand	21-06-2022	693	NaN
4	2	Jharkhand	12-06-2022	683	603.0
5	1	Jharkhand	22-06-2022	603	603.0
6	7	Maharastra	22-06-2022	603	603.0
7	8	Maharastra	22-06-2022	603	603.0
8	9	Maharastra	22-06-2022	603	603.0

By default, the window frame is:

`RANGE BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW` So the nth value may not exist yet for early rows.

Moving Average/Rolling Average/Rolling Mean in SQL

it is a smoothing technique for time series data. it removed the spikes/noise

In [28]:
```
%%sql
create table if not exists MovingAvg(Date date,UnitSales integer)
```

```
 * sqlite:///practice3.db
```

Out[28]: —

In [29]:
```
%%sql
insert into MovingAvg values
('28-05-2022',45),
('29-05-2022',36),
('30-05-2022',39),
('31-05-2022',45),
('01-06-2022',44),
('02-06-2022',35),
('03-06-2022',48),
('04-06-2022',45),
('05-06-2022',30)
```

```
 * sqlite:///practice3.db
```

Out[29]: —

In [30]:
```
%%sql
select * from MovingAvg
```

```
 * sqlite:///practice3.db
```

Out[30]:

	Date	UnitSales
0	28-05-2022	45
1	29-05-2022	36
2	30-05-2022	39
3	31-05-2022	45
4	01-06-2022	44
5	02-06-2022	35
6	03-06-2022	48
7	04-06-2022	45
8	05-06-2022	30
9	28-05-2022	45
10	29-05-2022	36
11	30-05-2022	39
12	31-05-2022	45
13	01-06-2022	44
14	02-06-2022	35
15	03-06-2022	48
16	04-06-2022	45
17	05-06-2022	30

In [31]:
```
%%sql
select *,Avg(UnitSales) over (order by Date rows between 2 preceding and
from MovingAvg
```

```
* sqlite:///practice3.db
```

Out[31]:

	Date	UnitSales	ThreeDaysMovingAvg
0	01-06-2022	44	44.000000
1	01-06-2022	44	44.000000
2	02-06-2022	35	41.000000
3	02-06-2022	35	38.000000
4	03-06-2022	48	39.333333
5	03-06-2022	48	43.666667
6	04-06-2022	45	47.000000
7	04-06-2022	45	46.000000
8	05-06-2022	30	40.000000
9	05-06-2022	30	35.000000
10	28-05-2022	45	35.000000
11	28-05-2022	45	40.000000
12	29-05-2022	36	42.000000
13	29-05-2022	36	39.000000
14	30-05-2022	39	37.000000
15	30-05-2022	39	38.000000
16	31-05-2022	45	41.000000
17	31-05-2022	45	43.000000

In [32]:

```
%%sql
select *,Avg(UnitSales) over (rows between 2 preceding and current row) a
Avg(UnitSales) over (rows between 4 preceding and current row) as FiveDay
from MovingAvg
```

```
* sqlite:///practice3.db
```

Out[32]:

	Date	UnitSales	ThreeDaysMovingAvg	FiveDaysMovingAvg
0	28-05-2022	45	45.000000	45.00
1	29-05-2022	36	40.500000	40.50
2	30-05-2022	39	40.000000	40.00
3	31-05-2022	45	40.000000	41.25
4	01-06-2022	44	42.666667	41.80
5	02-06-2022	35	41.333333	39.80
6	03-06-2022	48	42.333333	42.20
7	04-06-2022	45	42.666667	43.40
8	05-06-2022	30	41.000000	40.40
9	28-05-2022	45	40.000000	40.60
10	29-05-2022	36	37.000000	40.80
11	30-05-2022	39	40.000000	39.00
12	31-05-2022	45	40.000000	39.00
13	01-06-2022	44	42.666667	41.80
14	02-06-2022	35	41.333333	39.80
15	03-06-2022	48	42.333333	42.20
16	04-06-2022	45	42.666667	43.40
17	05-06-2022	30	41.000000	40.40

In [33]:
```
%%sql
create table NewStocks(Date date, Close float);
```

```
 * sqlite:///practice3.db
(sqlite3.OperationalError) table NewStocks already exists
[SQL: create table NewStocks(Date date, Close float);]
(Background on this error at: https://sqlalche.me/e/20/e3q8)
```

In [34]:
```
%%sql
insert into NewStocks values
('28-05-2021',1103.5),
('31-05-2021',1125.65),
('01-06-2021',1100.9),
('02-06-2021',1124.05),
('03-06-2021',1120.7),
('04-06-2021',1128.7),
('07-06-2021',1111.1),
('08-06-2021',1114.45),
('09-06-2021',1158.35)
```

```
 * sqlite:///practice3.db
```

Out[34]: —

In [35]:
```
%%sql
select *,avg(Close) over (rows between 2 preceding and current row) as Th
from NewStocks
```

```
* sqlite:///practice3.db
```

Out[35]:

	Date	Close	ThreeDaysMovingAvg
0	28-05-2021	1103.50	1103.500000
1	31-05-2021	1125.65	1114.575000
2	01-06-2021	1100.90	1110.016667
3	02-06-2021	1124.05	1116.866667
4	03-06-2021	1120.70	1115.216667
5	04-06-2021	1128.70	1124.483333
6	07-06-2021	1111.10	1120.166667
7	08-06-2021	1114.45	1118.083333
8	09-06-2021	1158.35	1127.966667
9	28-05-2021	1103.50	1125.433333
10	31-05-2021	1125.65	1129.166667
11	01-06-2021	1100.90	1110.016667
12	02-06-2021	1124.05	1116.866667
13	03-06-2021	1120.70	1115.216667
14	04-06-2021	1128.70	1124.483333
15	07-06-2021	1111.10	1120.166667
16	08-06-2021	1114.45	1118.083333
17	09-06-2021	1158.35	1127.966667

Lead and Lag in SQL

In [36]:
```
%%sql
create table if not exists TrainJourney(TrainNumber integer, Station text
```

```
* sqlite:///practice3.db
```

Out[36]: —

In [37]:
```
%%sql
insert into TrainJourney values
(22863, 'Howrah', '10:50:00'),
(22863, 'Kharagpur', '12:30:00'),
(22863, 'Balasore', '13:52:00'),
(22863, 'Cuttack', '15:47:00'),
(22863, 'Bhubaneswar', '16:25:00'),
(12262, 'Howrah', '05:45:00'),
(12262, 'Tatanagar', '09:00:00'),
(12262, 'Bilaspur', '15:05:00'),
(12262, 'Raipur', '16:37:00'),
(12262, 'Nagpur', '20:55:00')
```

```
* sqlite:///practice3.db
```

Out[37]: —

In [38]:
```
%%sql
select *,lead(Time) over (partition by TrainNumber order by Time) as Next
from TrainJourney
```

```
* sqlite:///practice3.db
```

Out[38]:

	TrainNumber	Station	Time	NextStationArrivalTime
0	12262	Howrah	05:45:00	05:45:00
1	12262	Howrah	05:45:00	09:00:00
2	12262	Tatanagar	09:00:00	09:00:00
3	12262	Tatanagar	09:00:00	15:05:00
4	12262	Bilaspur	15:05:00	15:05:00
5	12262	Bilaspur	15:05:00	16:37:00
6	12262	Raipur	16:37:00	16:37:00
7	12262	Raipur	16:37:00	20:55:00
8	12262	Nagpur	20:55:00	20:55:00
9	12262	Nagpur	20:55:00	None
10	22863	Howrah	10:50:00	10:50:00
11	22863	Howrah	10:50:00	12:30:00
12	22863	Kharagpur	12:30:00	12:30:00
13	22863	Kharagpur	12:30:00	13:52:00
14	22863	Balasore	13:52:00	13:52:00
15	22863	Balasore	13:52:00	15:47:00
16	22863	Cuttack	15:47:00	15:47:00
17	22863	Cuttack	15:47:00	16:25:00
18	22863	Bhubaneswar	16:25:00	16:25:00
19	22863	Bhubaneswar	16:25:00	None

In [39]:
```
%%sql
select *,lead(Time) over (partition by TrainNumber order by Time)-Time as
from TrainJourney
```

```
* sqlite:///practice3.db
```

Out[39]:

	TrainNumber	Station	Time	TimeToNextStation
0	12262	Howrah	05:45:00	0.0
1	12262	Howrah	05:45:00	4.0
2	12262	Tatanagar	09:00:00	0.0
3	12262	Tatanagar	09:00:00	6.0
4	12262	Bilaspur	15:05:00	0.0
5	12262	Bilaspur	15:05:00	1.0
6	12262	Raipur	16:37:00	0.0
7	12262	Raipur	16:37:00	4.0
8	12262	Nagpur	20:55:00	0.0
9	12262	Nagpur	20:55:00	NaN
10	22863	Howrah	10:50:00	0.0
11	22863	Howrah	10:50:00	2.0
12	22863	Kharagpur	12:30:00	0.0
13	22863	Kharagpur	12:30:00	1.0
14	22863	Balasore	13:52:00	0.0
15	22863	Balasore	13:52:00	2.0
16	22863	Cuttack	15:47:00	0.0
17	22863	Cuttack	15:47:00	1.0
18	22863	Bhubaneswar	16:25:00	0.0
19	22863	Bhubaneswar	16:25:00	NaN

here output is not correct but query is correct.

Because SQLite has no native TIME data type like PostgreSQL or Oracle.

In [40]:
```
%%sql
CREATE TABLE IF NOT EXISTS NewTrainJourney (
    TrainNumber INTEGER,
    Station TEXT,
    Time TEXT
);
```

```
* sqlite:///practice3.db
```

Out[40]: —

In [41]:
```
%%sql
INSERT INTO NewTrainJourney VALUES
(22863, 'Howrah', '10:50:00'),
(22863, 'Kharagpur', '12:30:00'),
(22863, 'Balasore', '13:52:00'),
(22863, 'Cuttack', '15:47:00'),
(22863, 'Bhubaneswar', '16:25:00'),
(12262, 'Howrah', '05:45:00'),
```

```
(12262, 'Tatanagar', '09:00:00'),
(12262, 'Bilaspur', '15:05:00'),
(12262, 'Raipur', '16:37:00'),
(12262, 'Nagpur', '20:55:00');
```

```
* sqlite:///practice3.db
```

Out[41]: —

In [42]:
```
%%sql
SELECT
    TrainNumber,
    Station,
    Time,
    (
        strftime('%s', '1970-01-01 ' || LEAD(Time) OVER (
            PARTITION BY TrainNumber
            ORDER BY Time
        ))
        -
        strftime('%s', '1970-01-01 ' || Time)
    ) / 60 AS TimeToNextStation_Minutes
FROM NewTrainJourney;
```

```
* sqlite:///practice3.db
```

Out[42]:

	TrainNumber	Station	Time	TimeToNextStation_Minutes
0	12262	Howrah	05:45:00	0.0
1	12262	Howrah	05:45:00	195.0
2	12262	Tatanagar	09:00:00	0.0
3	12262	Tatanagar	09:00:00	365.0
4	12262	Bilaspur	15:05:00	0.0
5	12262	Bilaspur	15:05:00	92.0
6	12262	Raipur	16:37:00	0.0
7	12262	Raipur	16:37:00	258.0
8	12262	Nagpur	20:55:00	0.0
9	12262	Nagpur	20:55:00	NaN
10	22863	Howrah	10:50:00	0.0
11	22863	Howrah	10:50:00	100.0
12	22863	Kharagpur	12:30:00	0.0
13	22863	Kharagpur	12:30:00	82.0
14	22863	Balasore	13:52:00	0.0
15	22863	Balasore	13:52:00	115.0
16	22863	Cuttack	15:47:00	0.0
17	22863	Cuttack	15:47:00	38.0
18	22863	Bhubaneswar	16:25:00	0.0
19	22863	Bhubaneswar	16:25:00	NaN

In [43]:

```
%%sql
SELECT
    TrainNumber,
    Station,
    Time,
    time(
        strftime('%s', '1970-01-01 ' || LEAD(Time) OVER (
            PARTITION BY TrainNumber
            ORDER BY Time
        ))
        -
        strftime('%s', '1970-01-01 ' || Time),
        'unixepoch'
    ) AS TimeToNextStation
FROM NewTrainJourney;
```

```
* sqlite:///practice3.db
```

Out[43]:

	TrainNumber	Station	Time	TimeToNextStation
0	12262	Howrah	05:45:00	00:00:00
1	12262	Howrah	05:45:00	03:15:00
2	12262	Tatanagar	09:00:00	00:00:00
3	12262	Tatanagar	09:00:00	06:05:00
4	12262	Bilaspur	15:05:00	00:00:00
5	12262	Bilaspur	15:05:00	01:32:00
6	12262	Raipur	16:37:00	00:00:00
7	12262	Raipur	16:37:00	04:18:00
8	12262	Nagpur	20:55:00	00:00:00
9	12262	Nagpur	20:55:00	None
10	22863	Howrah	10:50:00	00:00:00
11	22863	Howrah	10:50:00	01:40:00
12	22863	Kharagpur	12:30:00	00:00:00
13	22863	Kharagpur	12:30:00	01:22:00
14	22863	Balasore	13:52:00	00:00:00
15	22863	Balasore	13:52:00	01:55:00
16	22863	Cuttack	15:47:00	00:00:00
17	22863	Cuttack	15:47:00	00:38:00
18	22863	Bhubaneswar	16:25:00	00:00:00
19	22863	Bhubaneswar	16:25:00	None

Question

In [44]:
```
%%sql
create table if not exists Players(Player text, Year integer, Runs intege
```

```
* sqlite:///practice3.db
```

Out[44]: —

In [45]:
```
%%sql
insert into Players values
('Virat', 2008, 159),
('Virat', 2009, 325),
('Rohit', 2010, 225)
```

```
* sqlite:///practice3.db
```

Out[45]: —

In [46]:
```
%sql select * from Players;
```

```
* sqlite:///practice3.db
```

Out[46]:

	Player	Year	Runs
0	Virat	2008	159
1	Virat	2009	325
2	Rohit	2010	225
3	Virat	2008	159
4	Virat	2009	325
5	Rohit	2010	225

A. Total runs scored by Virat and Rohit

In [47]:
```
%%sql
select Player, sum(Runs) as TotalRuns from Players group by Player;
```

```
* sqlite:///practice3.db
```

Out[47]:

	Player	TotalRuns
0	Rohit	450
1	Virat	968

B. Which year scored what percent of runs

In [48]:
```
%%sql
select *,sum(Runs) over (partition by Player) as TotalRuns from Players
```

```
* sqlite:///practice3.db
```

Out[48]:

	Player	Year	Runs	TotalRuns
0	Rohit	2010	225	450
1	Rohit	2010	225	450
2	Virat	2008	159	968
3	Virat	2009	325	968
4	Virat	2008	159	968
5	Virat	2009	325	968

In [49]:
```
%%sql
select *,cast(Runs as float)/(sum(Runs) over (partition by Player))*100 |
```

```
* sqlite:///practice3.db
```

Out[49]:

	Player	Year	Runs	PercentageRuns
0	Rohit	2010	225	50.00000
1	Rohit	2010	225	50.00000
2	Virat	2008	159	16.42562
3	Virat	2009	325	33.57438
4	Virat	2008	159	16.42562
5	Virat	2009	325	33.57438

In [50]:
```
%%sql
SELECT
    *,
    (
        CAST(Runs AS REAL)
        / SUM(Runs) OVER (PARTITION BY Player) * 100
    ) || '%' AS PercentageRuns
FROM Players;
```

```
* sqlite:///practice3.db
```

Out[50]:

	Player	Year	Runs	PercentageRuns
0	Rohit	2010	225	50.0%
1	Rohit	2010	225	50.0%
2	Virat	2008	159	16.4256198347107%
3	Virat	2009	325	33.5743801652893%
4	Virat	2008	159	16.4256198347107%
5	Virat	2009	325	33.5743801652893%

In SQLite:

- || (string concatenation) has higher precedence.
- / and * have lower precedence

C. In How many years, they scored runs less than previous year

In [51]:
```
%%sql
select *,lag(Runs) over (partition by Player order by Year) as PreviousYe
from Players
```

```
* sqlite:///practice3.db
```

Out[51]:

	Player	Year	Runs	PreviousYearRuns
0	Rohit	2010	225	NaN
1	Rohit	2010	225	225.0
2	Virat	2008	159	NaN
3	Virat	2008	159	159.0
4	Virat	2009	325	159.0
5	Virat	2009	325	325.0

In [52]:
```
%%sql
select *, case when Runs  < lag(Runs) over (partition by Player order by
then 1 else 0 end as DecreaseFlag
from Players
```

```
* sqlite:///practice3.db
```

Out[52]:

	Player	Year	Runs	DecreaseFlag
0	Rohit	2010	225	0
1	Rohit	2010	225	0
2	Virat	2008	159	0
3	Virat	2008	159	0
4	Virat	2009	325	0
5	Virat	2009	325	0

In [53]:
```
%%sql
select P.Player, sum(DecreaseFlag) as TotalDecreaseYears from
(select *, case when Runs  < lag(Runs) over (partition by Player order by
then 1 else 0 end as DecreaseFlag
from Players) P group by P.Player
```

```
* sqlite:///practice3.db
```

Out[53]:

	Player	TotalDecreaseYears
0	Rohit	0
1	Virat	0

D. Count number of years in which Rohit has scored more than Virat

In [54]:
```
%%sql
select *,Runs-lead(Runs) over (partition by Year order by Player) as diff
from Players
```

```
* sqlite:///practice3.db
```

Out[54]:

	Player	Year	Runs	diff
0	Virat	2008	159	0.0
1	Virat	2008	159	NaN
2	Virat	2009	325	0.0
3	Virat	2009	325	NaN
4	Rohit	2010	225	0.0
5	Rohit	2010	225	NaN

Do it yourself now from here

- order of execution: where --> group by --> having --> order by --> limit.
- cross join works even there is no common column by using cartestian product. For example,

  `select a.*,b.* from table a, table2 b.`

- Non-Equi Join is used when we have to apply join condition other than "=" like <,>,!=

ISNULL and NULLIF in SQL

In [55]:
```
%%sql
create table if not exists marks(id integer primary key,
maths integer,english integer, physics integer,chemistry integer,
computer integer);
```

```
* sqlite:///practice3.db
```

Out[55]: —

In [56]:
```
%%sql
insert into marks values
(1,34,31,NULL,12,36),
(2,NULL,NULL,45,NULL,35)
```

```
 * sqlite:///practice3.db
(sqlite3.IntegrityError) UNIQUE constraint failed: marks.id
[SQL: insert into marks values
(1,34,31,NULL,12,36),
(2,NULL,NULL,45,NULL,35)]
(Background on this error at: https://sqlalche.me/e/20/gkpj)
```

In [57]:
```
%sql select * from marks
```

```
* sqlite:///practice3.db
```

Out[57]:

	id	maths	english	physics	chemistry	computer
0	1	34.0	31.0	NaN	12.0	36
1	2	NaN	NaN	45.0	NaN	35

we need to find sum of all subjects for each id by replacing null with zero

ISNULL() is not a valid function in SQLite.
ISNULL() works in SQL Server, but SQLite uses IFNULL() or COALESCE() instead.

In [58]:
```
%%sql
SELECT *,
       IFNULL(maths, 0) +
       IFNULL(english, 0) +
       IFNULL(physics, 0) +
       IFNULL(chemistry, 0) +
       IFNULL(computer, 0) AS total_marks
FROM marks;
```

```
* sqlite:///practice3.db
```

Out[58]:

	id	maths	english	physics	chemistry	computer	total_marks
0	1	34.0	31.0	NaN	12.0	36	113
1	2	NaN	NaN	45.0	NaN	35	80

opposite of isnull is nullif() --> replace non null to null

In [59]:
```
%%sql
select id, nullif(maths,267) from
(select id, ifnull(maths,0)+ifnull(english,0)+ifnull(physics,0)+ifnull(ch
from marks) a
```

```
 * sqlite:///practice3.db
(sqlite3.OperationalError) no such column: maths
[SQL: select id, nullif(maths,267) from
(select id, ifnull(maths,0)+ifnull(english,0)+ifnull(physics,0)+ifnull(che
mistry,0)+ifnull(computer,0)
from marks) a]
(Background on this error at: https://sqlalche.me/e/20/e3q8)
```

In [60]:
```
%%sql
SELECT id,
       NULLIF(total_marks, 267) AS total_marks_after_nullif
FROM (
    SELECT id,
           IFNULL(maths,0) +
           IFNULL(english,0) +
           IFNULL(physics,0) +
           IFNULL(chemistry,0) +
           IFNULL(computer,0) AS total_marks
    FROM marks
) a;
```

```
* sqlite:///practice3.db
```

Out[60]:

	id	total_marks_after_nullif
0	1	113
1	2	80

In [61]:
```
%%sql
SELECT id,
       NULLIF(total_marks, 113) AS total_marks_after_nullif
FROM (
    SELECT id,
           IFNULL(maths,0) +
           IFNULL(english,0) +
           IFNULL(physics,0) +
           IFNULL(chemistry,0) +
           IFNULL(computer,0) AS total_marks
    FROM marks
) a;
```

```
* sqlite:///practice3.db
```

Out[61]:

	id	total_marks_after_nullif
0	1	NaN
1	2	80.0

How NULLIF works here.

- If total_marks = 267 → result is NULL.
- Otherwise → result is total_marks.

In [62]:
```
%%sql
SELECT id,
       CASE
           WHEN total_marks IN (113, 120, 80) THEN NULL
           ELSE total_marks
       END AS total_marks_after_nullif
FROM (
    SELECT id,
           IFNULL(maths,0)+IFNULL(english,0)+IFNULL(physics,0)+
           IFNULL(chemistry,0)+IFNULL(computer,0) AS total_marks
    FROM marks
) a;
```

```
* sqlite:///practice3.db
```

Out[62]:

	id	total_marks_after_nullif
0	1	None
1	2	None

Joins

In [63]:
```
%%sql
create table if not exists JoinsDemo1(id integer);
create table if not exists JoinsDemo2(id integer);
```

```
* sqlite:///practice3.db
```

Out[63]: —

In [64]:
```
%%sql
insert into JoinsDemo1 values (1),(1),(2),(3),(4),(3),(NULL),(NULL);
insert into JoinsDemo2 values (1),(2),(3),(2),(NULL);
```

```
 * sqlite:///practice3.db
```

Out[64]: —

In [65]:
```
%%sql
SELECT * from JoinsDemo1;
select * from JoinsDemo2;
```

```
 * sqlite:///practice3.db
```

Out[65]:

	id
0	1.0
1	2.0
2	3.0
3	2.0
4	NaN
5	1.0
6	2.0
7	3.0
8	2.0
9	NaN

%%sql executes all statements, but displays results only for the LAST SELECT statement in the cell. it's how the Jupyter SQL magic is designed.

In [66]:
```
from IPython.display import display

res1 = %sql SELECT * FROM JoinsDemo1
res2 = %sql SELECT * FROM JoinsDemo2

display(res1)
display(res2)
```

```
 * sqlite:///practice3.db
 * sqlite:///practice3.db
```

	id
0	1.0
1	1.0
2	2.0
3	3.0
4	4.0
5	3.0
6	NaN
7	NaN
8	1.0
9	1.0
10	2.0
11	3.0
12	4.0
13	3.0
14	NaN
15	NaN

	id
0	1.0
1	2.0
2	3.0
3	2.0
4	NaN
5	1.0
6	2.0
7	3.0
8	2.0
9	NaN

In [67]:
```
%%sql
select * from JoinsDemo1 inner join JoinsDemo2
on JoinsDemo1.id = JoinsDemo2.id;
```

```
* sqlite:///practice3.db
```

Out[67]:

	id	id
0	1	1
1	1	1
2	1	1
3	1	1
4	2	2
5	2	2
6	2	2
7	2	2
8	3	3
9	3	3
10	3	3
11	3	3
12	1	1
13	1	1
14	1	1
15	1	1
16	2	2
17	2	2
18	2	2
19	2	2
20	3	3
21	3	3
22	3	3
23	3	3

In [68]:
```
%%sql
select * from JoinsDemo1 inner join JoinsDemo2
on JoinsDemo1.id = 1;
```

```
* sqlite:///practice3.db
```

Out[68]:

	id	id
0	1	1.0
1	1	1.0
2	1	1.0
3	1	1.0
4	1	2.0
5	1	2.0
6	1	2.0
7	1	2.0
8	1	3.0
9	1	3.0
10	1	3.0
11	1	3.0
12	1	2.0
13	1	2.0
14	1	2.0
15	1	2.0
16	1	NaN
17	1	NaN
18	1	NaN
19	1	NaN
20	1	1.0
21	1	1.0
22	1	1.0
23	1	1.0
24	1	2.0
25	1	2.0
26	1	2.0
27	1	2.0
28	1	3.0
29	1	3.0
30	1	3.0
31	1	3.0
32	1	2.0
33	1	2.0

	id	id
34	1	2.0
35	1	2.0
36	1	NaN
37	1	NaN
38	1	NaN
39	1	NaN

In [69]:
```
%%sql
SELECT * from JoinsDemo1 left join JoinsDemo2
on JoinsDemo1.id = JoinsDemo2.id;
```

```
* sqlite:///practice3.db
```

Out[69]:

	id	id
0	1.0	1.0
1	1.0	1.0
2	1.0	1.0
3	1.0	1.0
4	2.0	2.0
5	2.0	2.0
6	2.0	2.0
7	2.0	2.0
8	3.0	3.0
9	3.0	3.0
10	4.0	NaN
11	3.0	3.0
12	3.0	3.0
13	NaN	NaN
14	NaN	NaN
15	1.0	1.0
16	1.0	1.0
17	1.0	1.0
18	1.0	1.0
19	2.0	2.0
20	2.0	2.0
21	2.0	2.0
22	2.0	2.0
23	3.0	3.0
24	3.0	3.0
25	4.0	NaN
26	3.0	3.0
27	3.0	3.0
28	NaN	NaN
29	NaN	NaN

In [70]:
```
%%sql
SELECT * from JoinsDemo1 right join JoinsDemo2
on JoinsDemo1.id = JoinsDemo2.id;
```

```
* sqlite:///practice3.db
```

Out[70]:

	id	id
0	1.0	1.0
1	1.0	1.0
2	1.0	1.0
3	1.0	1.0
4	2.0	2.0
5	2.0	2.0
6	2.0	2.0
7	2.0	2.0
8	3.0	3.0
9	3.0	3.0
10	3.0	3.0
11	3.0	3.0
12	1.0	1.0
13	1.0	1.0
14	1.0	1.0
15	1.0	1.0
16	2.0	2.0
17	2.0	2.0
18	2.0	2.0
19	2.0	2.0
20	3.0	3.0
21	3.0	3.0
22	3.0	3.0
23	3.0	3.0
24	NaN	NaN
25	NaN	NaN

In [71]:
```
%%sql
SELECT * from JoinsDemo1 full outer join JoinsDemo2
on JoinsDemo1.id = JoinsDemo2.id;
```

```
* sqlite:///practice3.db
```

Out[71]:

	id	id
0	1.0	1.0
1	1.0	1.0
2	1.0	1.0
3	1.0	1.0
4	2.0	2.0
5	2.0	2.0
6	2.0	2.0
7	2.0	2.0
8	3.0	3.0
9	3.0	3.0
10	4.0	NaN
11	3.0	3.0
12	3.0	3.0
13	NaN	NaN
14	NaN	NaN
15	1.0	1.0
16	1.0	1.0
17	1.0	1.0
18	1.0	1.0
19	2.0	2.0
20	2.0	2.0
21	2.0	2.0
22	2.0	2.0
23	3.0	3.0
24	3.0	3.0
25	4.0	NaN
26	3.0	3.0
27	3.0	3.0
28	NaN	NaN
29	NaN	NaN
30	NaN	NaN
31	NaN	NaN

In [72]:
```
%%sql
SELECT * from JoinsDemo1 left join JoinsDemo2
```

```
on JoinsDemo1.id = JoinsDemo2.id

union all

SELECT * from JoinsDemo1 right join JoinsDemo2
on JoinsDemo1.id = JoinsDemo2.id
```

```
* sqlite:///practice3.db
```

Out[72]:

	id	id
0	1.0	1.0
1	1.0	1.0
2	1.0	1.0
3	1.0	1.0
4	2.0	2.0
5	2.0	2.0
6	2.0	2.0
7	2.0	2.0
8	3.0	3.0
9	3.0	3.0
10	4.0	NaN
11	3.0	3.0
12	3.0	3.0
13	NaN	NaN
14	NaN	NaN
15	1.0	1.0
16	1.0	1.0
17	1.0	1.0
18	1.0	1.0
19	2.0	2.0
20	2.0	2.0
21	2.0	2.0
22	2.0	2.0
23	3.0	3.0
24	3.0	3.0
25	4.0	NaN
26	3.0	3.0
27	3.0	3.0
28	NaN	NaN
29	NaN	NaN
30	1.0	1.0
31	1.0	1.0
32	1.0	1.0
33	1.0	1.0

	id	id
34	2.0	2.0
35	2.0	2.0
36	2.0	2.0
37	2.0	2.0
38	3.0	3.0
39	3.0	3.0
40	3.0	3.0
41	3.0	3.0
42	1.0	1.0
43	1.0	1.0
44	1.0	1.0
45	1.0	1.0
46	2.0	2.0
47	2.0	2.0
48	2.0	2.0
49	2.0	2.0
50	3.0	3.0
51	3.0	3.0
52	3.0	3.0
53	3.0	3.0
54	NaN	NaN
55	NaN	NaN

In [73]:
```
%%sql
SELECT * from JoinsDemo1 left join JoinsDemo2
on JoinsDemo1.id = JoinsDemo2.id

union

SELECT * from JoinsDemo1 right join JoinsDemo2
on JoinsDemo1.id = JoinsDemo2.id
```

```
* sqlite:///practice3.db
```

Out[73]:

	id	id
0	NaN	NaN
1	1.0	1.0
2	2.0	2.0
3	3.0	3.0
4	4.0	NaN

So, full outer join is not same as union of left and right joins

Question: Why union considers (NULL,NULL) and (NULL,NULL) as duplicate tuples but while joining tables, NULL can be interpretated as any value ?
Answer: UNION uses set semantics where NULLs are treated as equal for deduplication, while JOIN uses 3-valued logic where NULL never equals NULL.

In [74]:
```
%%sql
create table if not exists JoinsDemo3(id integer);
create table if not exists JoinsDemo4(id integer);
insert into JoinsDemo3 values (1),(1),(2),(3),(4),(3);
insert into JoinsDemo4 values (1),(2),(3),(2);
```

```
* sqlite:///practice3.db
```

Out[74]: —

In [75]:
```
%sql select * from JoinsDemo3
```

```
* sqlite:///practice3.db
```

Out[75]:

	id
0	1
1	1
2	2
3	3
4	4
5	3
6	1
7	1
8	2
9	3
10	4
11	3

In [76]:
```
%sql select * from JoinsDemo4
```

```
* sqlite:///practice3.db
```

Out[76]:

	id
0	1
1	2
2	3
3	2
4	1
5	2
6	3
7	2

In [77]:
```
%%sql
select * from JoinsDemo3 inner join JoinsDemo4
on JoinsDemo3.id = JoinsDemo4.id;
```

```
* sqlite:///practice3.db
```

Out[77]:

	id	id
0	1	1
1	1	1
2	1	1
3	1	1
4	2	2
5	2	2
6	2	2
7	2	2
8	3	3
9	3	3
10	3	3
11	3	3
12	1	1
13	1	1
14	1	1
15	1	1
16	2	2
17	2	2
18	2	2
19	2	2
20	3	3
21	3	3
22	3	3
23	3	3

In [78]:
```
%%sql
select * from JoinsDemo3 left join JoinsDemo4
on JoinsDemo3.id = JoinsDemo4.id;
```

```
* sqlite:///practice3.db
```

Out[78]:

	id	id
0	1	1.0
1	1	1.0
2	1	1.0
3	1	1.0
4	2	2.0
5	2	2.0
6	2	2.0
7	2	2.0
8	3	3.0
9	3	3.0
10	4	NaN
11	3	3.0
12	3	3.0
13	1	1.0
14	1	1.0
15	1	1.0
16	1	1.0
17	2	2.0
18	2	2.0
19	2	2.0
20	2	2.0
21	3	3.0
22	3	3.0
23	4	NaN
24	3	3.0
25	3	3.0

In [79]:
```
%%sql
select * from JoinsDemo3 right join JoinsDemo4
on JoinsDemo3.id = JoinsDemo4.id;
```

```
* sqlite:///practice3.db
```

Out[79]:

	id	id
0	1	1
1	1	1
2	1	1
3	1	1
4	2	2
5	2	2
6	2	2
7	2	2
8	3	3
9	3	3
10	3	3
11	3	3
12	1	1
13	1	1
14	1	1
15	1	1
16	2	2
17	2	2
18	2	2
19	2	2
20	3	3
21	3	3
22	3	3
23	3	3

In [80]:
```
%%sql
select * from JoinsDemo3 full outer join JoinsDemo4
on JoinsDemo3.id = JoinsDemo4.id;
```

```
* sqlite:///practice3.db
```

Out[80]:

	id	id
0	1	1.0
1	1	1.0
2	1	1.0
3	1	1.0
4	2	2.0
5	2	2.0
6	2	2.0
7	2	2.0
8	3	3.0
9	3	3.0
10	4	NaN
11	3	3.0
12	3	3.0
13	1	1.0
14	1	1.0
15	1	1.0
16	1	1.0
17	2	2.0
18	2	2.0
19	2	2.0
20	2	2.0
21	3	3.0
22	3	3.0
23	4	NaN
24	3	3.0
25	3	3.0

set difference

In [81]:
```
%%sql
select * from JoinsDemo3 left join JoinsDemo4
on JoinsDemo3.id = JoinsDemo4.id

except

select * from JoinsDemo3 right join JoinsDemo4
on JoinsDemo3.id = JoinsDemo4.id
```

```
* sqlite:///practice3.db
```

Out[81]:

	id	id
0	4	None

Difference between 2 timestamps in SQL

In [82]:
```
%%sql
CREATE TABLE IF NOT EXISTS TimeDiffDemo (
    id INTEGER PRIMARY KEY,
    start_time TEXT,
    end_time   TEXT
);
```

```
* sqlite:///practice3.db
```

Out[82]: —

SQLite stores timestamps as TEXT in ISO format.

In [83]:
```
%%sql
INSERT INTO TimeDiffDemo (start_time, end_time) VALUES
('2025-12-20 10:15:00', '2025-12-20 13:45:00'),
('2025-12-20 09:00:00', '2025-12-20 10:20:00'),
('2025-12-20 22:30:00', '2025-12-21 01:10:00');
```

```
* sqlite:///practice3.db
```

Out[83]: —

SQLite does not have TIMESTAMPDIFF, so we use julianday()

In [84]:
```
%%sql
SELECT id,
       start_time,
       end_time,
       ROUND((julianday(end_time) - julianday(start_time)) * 24, 2)
           AS diff_hours
FROM TimeDiffDemo;
```

```
* sqlite:///practice3.db
```

Out[84]:

	id	start_time	end_time	diff_hours
0	1	2025-12-20 10:15:00	2025-12-20 13:45:00	3.50
1	2	2025-12-20 09:00:00	2025-12-20 10:20:00	1.33
2	3	2025-12-20 22:30:00	2025-12-21 01:10:00	2.67
3	4	2025-12-20 10:15:00	2025-12-20 13:45:00	3.50
4	5	2025-12-20 09:00:00	2025-12-20 10:20:00	1.33
5	6	2025-12-20 22:30:00	2025-12-21 01:10:00	2.67

-- julianday() → difference in days

-- * 24 → convert to hours

In [85]:
```
%%sql
SELECT id,
```

```
       start_time,
       end_time,
       ROUND((julianday(end_time) - julianday(start_time)) * 24 * 60)
           AS diff_minutes
FROM TimeDiffDemo;
```

```
* sqlite:///practice3.db
```

Out[85]:

	id	start_time	end_time	diff_minutes
0	1	2025-12-20 10:15:00	2025-12-20 13:45:00	210.0
1	2	2025-12-20 09:00:00	2025-12-20 10:20:00	80.0
2	3	2025-12-20 22:30:00	2025-12-21 01:10:00	160.0
3	4	2025-12-20 10:15:00	2025-12-20 13:45:00	210.0
4	5	2025-12-20 09:00:00	2025-12-20 10:20:00	80.0
5	6	2025-12-20 22:30:00	2025-12-21 01:10:00	160.0

In [86]:

```
%%sql
SELECT id,
       start_time,
       end_time,
       CAST((julianday(end_time) - julianday(start_time)) * 24 AS INTEGER
           || ' hours '
           || CAST(((julianday(end_time) - julianday(start_time)) * 24 *
           || ' minutes' AS diff_hh_mm
FROM TimeDiffDemo;
```

```
* sqlite:///practice3.db
```

Out[86]:

	id	start_time	end_time	diff_hh_mm
0	1	2025-12-20 10:15:00	2025-12-20 13:45:00	3 hours 29 minutes
1	2	2025-12-20 09:00:00	2025-12-20 10:20:00	1 hours 19 minutes
2	3	2025-12-20 22:30:00	2025-12-21 01:10:00	2 hours 39 minutes
3	4	2025-12-20 10:15:00	2025-12-20 13:45:00	3 hours 29 minutes
4	5	2025-12-20 09:00:00	2025-12-20 10:20:00	1 hours 19 minutes
5	6	2025-12-20 22:30:00	2025-12-21 01:10:00	2 hours 39 minutes

In [87]:

```
%%sql
SELECT id,
       start_time,
       end_time,
       (strftime('%s', end_time) - strftime('%s', start_time)) / 3600 AS
       (strftime('%s', end_time) - strftime('%s', start_time)) / 60   AS
FROM TimeDiffDemo;
```

```
* sqlite:///practice3.db
```

Out[87]:

	id	start_time	end_time	hours	minutes
0	1	2025-12-20 10:15:00	2025-12-20 13:45:00	3	210
1	2	2025-12-20 09:00:00	2025-12-20 10:20:00	1	80
2	3	2025-12-20 22:30:00	2025-12-21 01:10:00	2	160
3	4	2025-12-20 10:15:00	2025-12-20 13:45:00	3	210
4	5	2025-12-20 09:00:00	2025-12-20 10:20:00	1	80
5	6	2025-12-20 22:30:00	2025-12-21 01:10:00	2	160

SQLite does not support TIMESTAMPDIFF. We calculate time difference using julianday() or Unix seconds via strftime('%s').

List Aggregate and String Aggregate in SQLite

In SQLite, there is no LISTAGG() or STRING_AGG() function (those exist in Oracle / PostgreSQL / SQL Server).

SQLite equivalent = GROUP_CONCAT()

In [88]:
```
%%sql
CREATE TABLE IF NOT EXISTS ListAggDemo (
    dept TEXT,
    emp_name TEXT
);
```

```
* sqlite:///practice3.db
```

Out[88]: —

In [89]:
```
%%sql
INSERT INTO ListAggDemo VALUES
('IT', 'Ankit'),
('IT', 'Ravi'),
('IT', 'Neha'),
('HR', 'Pooja'),
('HR', 'Amit'),
('Finance', 'Suresh');
```

```
* sqlite:///practice3.db
```

Out[89]: —

In [90]:
```
%%sql
SELECT dept,
       GROUP_CONCAT(emp_name) AS emp_list
FROM ListAggDemo
GROUP BY dept;
```

```
* sqlite:///practice3.db
```

Out[90]:

	dept	emp_list
0	Finance	Suresh,Suresh
1	HR	Pooja,Amit,Pooja,Amit
2	IT	Ankit,Ravi,Neha,Ankit,Ravi,Neha

Default separator is ","

STRING AGGREGATE with custom separator

In [91]:
```
%%sql
SELECT dept,
       GROUP_CONCAT(emp_name, ' | ') AS emp_list
FROM ListAggDemo
GROUP BY dept;
```

```
* sqlite:///practice3.db
```

Out[91]:

	dept	emp_list
0	Finance	Suresh | Suresh
1	HR	Pooja | Amit | Pooja | Amit
2	IT	Ankit | Ravi | Neha | Ankit | Ravi | Neha

SQLite does not support ORDER BY directly inside GROUP_CONCAT() You must use a subquery.

In [92]:
```
%%sql
SELECT dept,
       GROUP_CONCAT(emp_name, ', ') AS emp_list
FROM (
    SELECT dept, emp_name
    FROM ListAggDemo
    ORDER BY emp_name
)
GROUP BY dept;
```

```
* sqlite:///practice3.db
```

Out[92]:

	dept	emp_list
0	Finance	Suresh, Suresh
1	HR	Amit, Amit, Pooja, Pooja
2	IT	Ankit, Ankit, Neha, Neha, Ravi, Ravi

In [93]:
```
%%sql
SELECT dept,
       GROUP_CONCAT(DISTINCT emp_name) AS emp_list
FROM ListAggDemo
GROUP BY dept;
```

```
* sqlite:///practice3.db
```

Out[93]:

	dept	emp_list
0	Finance	Suresh
1	HR	Pooja,Amit
2	IT	Ankit,Ravi,Neha

In [94]:
```
%%sql
SELECT dept,
       COUNT(*) AS emp_count,
       GROUP_CONCAT(emp_name, ', ') AS emp_list
FROM ListAggDemo
GROUP BY dept;
```

```
* sqlite:///practice3.db
```

Out[94]:

	dept	emp_count	emp_list
0	Finance	2	Suresh, Suresh
1	HR	4	Pooja, Amit, Pooja, Amit
2	IT	6	Ankit, Ravi, Neha, Ankit, Ravi, Neha

Interview Questions

Question:

In [95]:
```
%%sql
CREATE TABLE IF NOT EXISTS Customers (
    CustomerID INT PRIMARY KEY,
    CustomerName VARCHAR(100),
    Age INT,
    State VARCHAR(50)
);
```

```
* sqlite:///practice3.db
```

Out[95]: —

In [96]:
```
%%sql
insert into Customers values
(1, 'Ram', 21, 'Jharkhand'),
(2, 'Shyam', 26, 'Bihar'),
(3, 'Raj', 38, 'Jharkhand'),
(4, 'Rahul', 29, 'Jharkhand'),
(5, 'Suresh', 40, 'Jharkhand'),
(6, 'Ramesh', 33, 'West Bengal');
```

```
 * sqlite:///practice3.db
(sqlite3.IntegrityError) UNIQUE constraint failed: Customers.CustomerID
[SQL: insert into Customers values
(1, 'Ram', 21, 'Jharkhand'),
(2, 'Shyam', 26, 'Bihar'),
(3, 'Raj', 38, 'Jharkhand'),
(4, 'Rahul', 29, 'Jharkhand'),
(5, 'Suresh', 40, 'Jharkhand'),
(6, 'Ramesh', 33, 'West Bengal');]
(Background on this error at: https://sqlalche.me/e/20/gkpj)
```

In [97]:
```
%sql select * from Customers;
```

```
* sqlite:///practice3.db
```

Out[97]:

	CustomerID	CustomerName	Age	State
0	1	Ram	21	Jharkhand
1	2	Shyam	26	Bihar
2	3	Raj	38	Jharkhand
3	4	Rahul	29	Jharkhand
4	5	Suresh	40	Jharkhand
5	6	Ramesh	33	West Bengal

In [98]:
```
%%sql
CREATE TABLE IF NOT EXISTS Orders (
    CustomerID INT,
    OrderID INT PRIMARY KEY,
    orderdate DATE,
    amount FLOAT);
```

```
* sqlite:///practice3.db
```

Out[98]: —

In [99]:
```
%%sql
insert into Orders values
(1,1,'2021-04-19',560),
(1,2,'2021-04-24',3824),
(2,3,'2021-05-01',613),
(3,4,'2021-05-03',1399),
(3,5,'2021-05-28',4391),
(3,6,'2021-06-04',2877),
(5,7,'2021-04-08',4748),
(6,8,'2021-03-16',3352),
(6,9,'2021-05-04',2072);
```

```
 * sqlite:///practice3.db
(sqlite3.IntegrityError) UNIQUE constraint failed: Orders.OrderID
[SQL: insert into Orders values
(1,1,'2021-04-19',560),
(1,2,'2021-04-24',3824),
(2,3,'2021-05-01',613),
(3,4,'2021-05-03',1399),
(3,5,'2021-05-28',4391),
(3,6,'2021-06-04',2877),
(5,7,'2021-04-08',4748),
(6,8,'2021-03-16',3352),
(6,9,'2021-05-04',2072);]
(Background on this error at: https://sqlalche.me/e/20/gkpj)
```

In [100…
```
%sql select * from Orders;
```

```
* sqlite:///practice3.db
```

Out[100…

	CustomerID	OrderID	orderdate	amount
0	1	1	2021-04-19	560.0
1	1	2	2021-04-24	3824.0
2	2	3	2021-05-01	613.0
3	3	4	2021-05-03	1399.0
4	3	5	2021-05-28	4391.0
5	3	6	2021-06-04	2877.0
6	5	7	2021-04-08	4748.0
7	6	8	2021-03-16	3352.0
8	6	9	2021-05-04	2072.0

A. Write a query to get customer name, count of orders purchased in april 2021 and may2021

In [101…

```
%%sql
select c.CustomerName, b.count from
(select a.CustomerID, count(a.OrderID) as count from (select * from order
a group by a.CustomerID) b inner join Customers c
on b.CustomerID = c.CustomerID;
```

```
* sqlite:///practice3.db
```

Out[101…

	CustomerName	count
0	Ram	2
1	Shyam	1
2	Raj	2
3	Suresh	1
4	Ramesh	1

B. write a query to get customer names who bought in May 2021 and are from Jharkhand

In [102…

```
%%sql
select distinct C. CustomerName from Customers C inner join Orders O on C
O.orderdate between '2021-05-01' and '2021-05-31'
where C.State='Jharkhand';
```

```
* sqlite:///practice3.db
```

Out[102…

	CustomerName
0	Raj

C. write a query to get customer name and their latest order information

In [103…

```
%%sql
select C.CustomerName, O.orderdate from Customers C inner join Orders O o
```

```
WHERE O.orderdate = (SELECT MAX(orderdate) FROM Orders WHERE CustomerID=C
```

```
* sqlite:///practice3.db
```

Out[103…

	CustomerName	orderdate
0	Ram	2021-04-24
1	Shyam	2021-05-01
2	Raj	2021-06-04
3	Suresh	2021-04-08
4	Ramesh	2021-05-04

Other way:

In [104…

```
%%sql
select C.CustomerName, b.orderdate from (select * from (select *,rank() o
from Orders O) a where a.amt_rank =1) b inner join Customers C on b.Custo
```

```
* sqlite:///practice3.db
```

Out[104…

	CustomerName	orderdate
0	Ram	2021-04-24
1	Shyam	2021-05-01
2	Raj	2021-06-04
3	Suresh	2021-04-08
4	Ramesh	2021-05-04

D. write a query to get top 2 customer id and name based on total transaction value for each month

In [105…

```
%%sql
select * from (select a.CustomerID, sum(a.amount) as total from (select *
when orderdate between '2021-05-01' and '2021-05-31' then 5
when orderdate between '2021-04-01' and '2021-04-30' then 4
when orderdate between '2021-06-01' and '2021-06-30' then 6
when orderdate between '2021-03-01' and '2021-03-31' then 3
end as month from orders) a group by a.month ) b order by b.total desc li
```

```
* sqlite:///practice3.db
```

Out[105…

	CustomerID	total
0	1	9132.0
1	2	8475.0

other way:

In [106…

```
%%sql
select * from (select *, dense_rank() over (order by b.total_amount desc)
from (select a.CustomerID,sum(a.amount) as total_amount from (SELECT *,st
FROM orders) a group by a.month) b) where rnk <=2;
```

```
* sqlite:///practice3.db
```

Out[106…

	CustomerID	total_amount	rnk
0	1	9132.0	1
1	2	8475.0	2

CTEs (Common Table Expression) in SQL:

CTEs (temporary table) is a small subset of the dataset for usability.

A Common Table Expression (CTE) in SQLite is a temporary named result set that you define using the WITH keyword and then use within a single SQL query. It helps make complex queries cleaner, more readable, and easier to maintain.

Think of a CTE as a temporary table that exists only during the execution of one query.

A CTE is NOT a physical table. It is a temporary named query that exists only during one SQL statement.

A TEMP TABLE is a real table stored in memory (or temp disk). It exists for the duration of the database connection.

A CTE is a temporary table that exists only during the execution of one SQL statement.

In [108…

```sql
%%sql
CREATE TABLE employees1 (
    emp_id INTEGER PRIMARY KEY,
    name TEXT,
    department TEXT,
    salary INTEGER
);

INSERT INTO employees1 (name, department, salary) VALUES
('Ankit', 'IT', 80000),
('Rahul', 'HR', 50000),
('Priya', 'IT', 70000),
('Neha', 'Finance', 60000);

select * from employees1;
```

```
* sqlite:///practice3.db
```

Out[108…

	emp_id	name	department	salary
0	1	Ankit	IT	80000
1	2	Rahul	HR	50000
2	3	Priya	IT	70000
3	4	Neha	Finance	60000

A View is a stored SQL query that behaves like a table but does not store data.

SQLite stores only the query definition, not the result.

In [110…
```sql
%%sql
CREATE VIEW it_employees1 AS
SELECT emp_id, name, salary
FROM employees1
WHERE department = 'IT';

SELECT * FROM it_employees;
```

```
* sqlite:///practice3.db
```

Out[110…

	emp_id	name	salary
0	1	Ankit	80000
1	3	Priya	70000

SQLite views are read-only unless triggers are used. So,
`INSERT INTO it_employees1 VALUES (5, 'Aman', 75000);` will not work

Key Points (Views).
- No data stored.
- Always shows latest data.
- Stored permanently.
- Read-only in SQLite (mostly).

A temporary table:
- Stores actual data.
- Exists only for the current session.
- Automatically deleted when connection closes.

In [111…
```sql
%%sql
CREATE TEMP TABLE temp_high_salary (
    emp_id INTEGER,
    name TEXT,
    salary INTEGER
);

INSERT INTO temp_high_salary
SELECT emp_id, name, salary
FROM employees
WHERE salary > 60000;

SELECT * FROM temp_high_salary;
```

```
* sqlite:///practice3.db
```

Out[111…

	emp_id	name	salary
0	1	Ankit	80000
1	3	Priya	70000

What is a CTE?

A CTE is a temporary result set used only inside one query.

It exists only during query execution.

In [112…
```
%%sql
WITH high_salary AS (
    SELECT emp_id, name, salary
    FROM employees
    WHERE salary > 60000
)
SELECT * FROM high_salary;
```

```
* sqlite:///practice3.db
```

Out[112…

	emp_id	name	salary
0	1	Ankit	80000
1	3	Priya	70000

Cannot Reuse CTE

In [113…
```
%%sql
SELECT * FROM high_salary;
```

```
 * sqlite:///practice3.db
(sqlite3.OperationalError) no such table: high_salary
[SQL: SELECT * FROM high_salary;]
(Background on this error at: https://sqlalche.me/e/20/e3q8)
```

Recursive CTE (Supported in SQLite)

In [114…
```
%%sql
WITH RECURSIVE numbers(n) AS (
    SELECT 1
    UNION ALL
    SELECT n + 1 FROM numbers WHERE n < 5
)
SELECT * FROM numbers;
```

```
* sqlite:///practice3.db
```

Out[114…

	n
0	1
1	2
2	3
3	4
4	5

Key Points (CTEs):

- No storage.
- Query-only scope.
- Improves readability.
- Excellent for recursion.

Interview Question:

In [115…
```sql
%%sql
DROP TABLE IF EXISTS users;

CREATE TABLE users (
    voter_id INTEGER PRIMARY KEY,
    signup_date DATE
);

INSERT INTO users (voter_id, signup_date) VALUES
(1, '2009-09-22'),
(2, '2011-09-10'),
(3, '2015-09-21');

SELECT * FROM users;
```

```
* sqlite:///practice3.db
```

Out[115…

	voter_id	signup_date
0	1	2009-09-22
1	2	2011-09-10
2	3	2015-09-21

In [116…
```sql
%%sql
DROP TABLE IF EXISTS transactions;

CREATE TABLE transactions (
    transaction_id INTEGER PRIMARY KEY,
    voter_id INTEGER,
    created_at DATE,
    updated_at DATE,
    status TEXT,
    amount INTEGER
);

INSERT INTO transactions (transaction_id, voter_id, created_at, updated_a
(1, 1, '2017-04-19','2017-04-21', 'fail', 105),
(2, 3, '2019-12-18','2019-12-19', 'success', 215),
(3, 2, '2020-02-20','2020-07-23', 'fail', 436);

SELECT * FROM transactions;
```

```
* sqlite:///practice3.db
```

Out[116…

	transaction_id	voter_id	created_at	updated_at	status	amount
0	1	1	2017-04-19	2017-04-21	fail	105
1	2	3	2019-12-18	2019-12-19	success	215
2	3	2	2020-02-20	2020-07-23	fail	436

A. write a query to find all the transactions done by the most recently signed user

In [117…
```sql
%%sql
select b.* from (select * from users where signup_date = (select max(sign
```

```
inner join transactions b on a.voter_id=b.voter_id
```

```
* sqlite:///practice3.db
```

Out[117…

	transaction_id	voter_id	created_at	updated_at	status	amount
0	2	3	2019-12-18	2019-12-19	success	215

write a query to find transaction_ids of second highest amount transaction done by all users

In [118…

```
%%sql
select a.transaction_id from (select *,rank() over (partition by voter_id
where a.rnk=2
```

```
* sqlite:///practice3.db
```

Out[118… —

write a query to add a column(cumulative_amount) to transaction done by a user at every transaction_id (without any window function)

-- Without window function

In [119…

```
%%sql
select a.transaction_id,a.voter_id, a.updated_at,a.amount,sum(a.amt) cumu
(select a.*,b.amount amt from transactions a join transactions b on a.tra
and a.voter_id =b.voter_id) a
group by a.transaction_id,a.voter_id,a.updated_at,a.amount
```

```
* sqlite:///practice3.db
```

Out[119…

	transaction_id	voter_id	updated_at	amount	cumulative_amount
0	1	1	2017-04-21	105	105
1	2	3	2019-12-19	215	215
2	3	2	2020-07-23	436	436

-- With window function

In [120…

```
%%sql
select *, sum(amount) over(partition by voter_id order by transaction_id
unbounded preceding and current row) cumulative_amount from transactions;
```

```
* sqlite:///practice3.db
```

Out[120…

	transaction_id	voter_id	created_at	updated_at	status	amount	cumulative_
0	1	1	2017-04-19	2017-04-21	fail	105	
1	3	2	2020-02-20	2020-07-23	fail	436	
2	2	3	2019-12-18	2019-12-19	success	215	

Pivots in SQL:

comes from pivottable in excel

SQLite does not have a built-in PIVOT keyword (unlike SQL Server / Oracle), so pivoting is done using CASE WHEN + aggregate functions

In [121…
```
%%sql
CREATE TABLE IF NOT EXISTS sales (
    sale_id     INTEGER PRIMARY KEY,
    sale_date   DATE,
    product     TEXT,
    region      TEXT,
    amount      INTEGER
);

INSERT INTO sales (sale_date, product, region, amount) VALUES
('2024-01-01', 'Laptop', 'North', 800),
('2024-01-02', 'Laptop', 'South', 750),
('2024-01-03', 'Mobile', 'North', 500),
('2024-01-04', 'Mobile', 'South', 450),
('2024-01-05', 'Tablet', 'North', 300),
('2024-01-06', 'Tablet', 'South', 280),
('2024-01-07', 'Laptop', 'North', 820),
('2024-01-08', 'Mobile', 'South', 470);
```

```
* sqlite:///practice3.db
```

Out[121…

Normal (Unpivoted) View:

In [122…
```
%%sql
SELECT * FROM sales;
```

```
* sqlite:///practice3.db
```

Out [122…

	sale_id	sale_date	product	region	amount
0	1	2024-01-01	Laptop	North	800
1	2	2024-01-02	Laptop	South	750
2	3	2024-01-03	Mobile	North	500
3	4	2024-01-04	Mobile	South	450
4	5	2024-01-05	Tablet	North	300
5	6	2024-01-06	Tablet	South	280
6	7	2024-01-07	Laptop	North	820
7	8	2024-01-08	Mobile	South	470
8	9	2024-01-01	Laptop	North	800
9	10	2024-01-02	Laptop	South	750
10	11	2024-01-03	Mobile	North	500
11	12	2024-01-04	Mobile	South	450
12	13	2024-01-05	Tablet	North	300
13	14	2024-01-06	Tablet	South	280
14	15	2024-01-07	Laptop	North	820
15	16	2024-01-08	Mobile	South	470

This is row-based data, not pivoted.

A pivot converts rows into columns.

👉 Example Goal:

Rows → Products.
Columns → Regions.
Values → Total Sales Amount.

Basic Pivot Using CASE WHEN:

In [123…

```
%%sql
SELECT
    product,
    SUM(CASE WHEN region = 'North' THEN amount ELSE 0 END) AS north_sales
    SUM(CASE WHEN region = 'South' THEN amount ELSE 0 END) AS south_sales
FROM sales
GROUP BY product;
```

```
* sqlite:///practice3.db
```

Out[123…

	product	north_sales	south_sales
0	Laptop	3240	1500
1	Mobile	1000	1840
2	Tablet	600	560

Pivot with COUNT Instead of SUM:

Count number of sales per region per product

In [124…

```
%%sql
SELECT
    product,
    COUNT(CASE WHEN region = 'North' THEN 1 END) AS north_orders,
    COUNT(CASE WHEN region = 'South' THEN 1 END) AS south_orders
FROM sales
GROUP BY product;
```

```
* sqlite:///practice3.db
```

Out[124…

	product	north_orders	south_orders
0	Laptop	4	2
1	Mobile	2	4
2	Tablet	2	2

Pivot by Month (Date-Based Pivot)

Step 1: Extract Month

In [125…

```
%%sql
SELECT
    strftime('%m', sale_date) AS month,
    amount
FROM sales;
```

```
* sqlite:///practice3.db
```

Out[125…

	month	amount
0	01	800
1	01	750
2	01	500
3	01	450
4	01	300
5	01	280
6	01	820
7	01	470
8	01	800
9	01	750
10	01	500
11	01	450
12	01	300
13	01	280
14	01	820
15	01	470

Step 2: Monthly Sales Pivot

In [126…

```
%%sql
SELECT
    product,
    SUM(CASE WHEN strftime('%m', sale_date) = '01' THEN amount ELSE 0 END
FROM sales
GROUP BY product;
```

```
* sqlite:///practice3.db
```

Out[126…

	product	jan_sales
0	Laptop	4740
1	Mobile	2840
2	Tablet	1160

Multi-Column Pivot (Region × Product):

In [127…

```
%%sql
SELECT
    region,
    SUM(CASE WHEN product = 'Laptop' THEN amount ELSE 0 END) AS laptop_sa
    SUM(CASE WHEN product = 'Mobile' THEN amount ELSE 0 END) AS mobile_sa
    SUM(CASE WHEN product = 'Tablet' THEN amount ELSE 0 END) AS tablet_sa
FROM sales
GROUP BY region;
```

```
* sqlite:///practice3.db
```

Out[127…

	region	laptop_sales	mobile_sales	tablet_sales
0	North	3240	1000	600
1	South	1500	1840	560

Pivot Using AVG, MIN, MAX:

Average sale amount per product per region

In [128…

```
%%sql
SELECT
    product,
    AVG(CASE WHEN region = 'North' THEN amount END) AS avg_north,
    AVG(CASE WHEN region = 'South' THEN amount END) AS avg_south
FROM sales
GROUP BY product;
```

```
* sqlite:///practice3.db
```

Out[128…

	product	avg_north	avg_south
0	Laptop	810.0	750.0
1	Mobile	500.0	460.0
2	Tablet	300.0	280.0

SQLite does NOT support dynamic pivots automatically.

Interview Question(Recursive CTEs):

In [129…

```
%%sql
create table if not exists orders1(
    order_id text,
    product_id text,
    quantity int
);
insert into orders1 values
('ORD1','PRD1',5),
('ORD2','PRD2',1),
('ORD3','PRD3',3);

SELECT * FROM orders1;
```

```
* sqlite:///practice3.db
```

Out[129…

	order_id	product_id	quantity
0	ORD1	PRD1	5
1	ORD2	PRD2	1
2	ORD3	PRD3	3
3	ORD1	PRD1	5
4	ORD2	PRD2	1
5	ORD3	PRD3	3

-- Write a SQL query which will explode data into single unit level records.

Output we want:

ORD1 | PRD1 | 1.
ORD1 | PRD1 | 1.
ORD1 | PRD1 | 1.
ORD1 | PRD1 | 1.
ORD1 | PRD1 | 1.
ORD2 | PRD2 | 1.
ORD3 | PRD3 | 1.
ORD3 | PRD3 | 1.
ORD3 | PRD3 | 1 (opposite of group by)

In [130…

```
%%sql
WITH RECURSIVE numbers(n) AS (
    SELECT 1          -- start value
    UNION ALL
    SELECT n + 1
    FROM numbers
    WHERE n < 10      -- stop condition
)
SELECT * FROM numbers;
```

```
* sqlite:///practice3.db
```

Out[130…

	n
0	1
1	2
2	3
3	4
4	5
5	6
6	7
7	8
8	9
9	10

This behaves like:

```
for n in range(1, 11):
    print(n)
```

Now, we can use this to explode data:

In [131…

```
%%sql
WITH RECURSIVE exploded_orders AS (

    -- Anchor member: start with original quantity
    SELECT
        order_id,
        product_id,
        quantity
    FROM orders1

    UNION ALL

    -- Recursive member: reduce quantity by 1
    SELECT
        order_id,
        product_id,
        quantity - 1
    FROM exploded_orders
    WHERE quantity > 1
)

SELECT
    order_id,
    product_id,
    1 AS quantity
FROM exploded_orders
ORDER BY order_id;
```

```
* sqlite:///practice3.db
```

Out[131…

	order_id	product_id	quantity
0	ORD1	PRD1	1
1	ORD1	PRD1	1
2	ORD1	PRD1	1
3	ORD1	PRD1	1
4	ORD1	PRD1	1
5	ORD1	PRD1	1
6	ORD1	PRD1	1
7	ORD1	PRD1	1
8	ORD1	PRD1	1
9	ORD1	PRD1	1
10	ORD2	PRD2	1
11	ORD2	PRD2	1
12	ORD3	PRD3	1
13	ORD3	PRD3	1
14	ORD3	PRD3	1
15	ORD3	PRD3	1
16	ORD3	PRD3	1
17	ORD3	PRD3	1

Interview Question

In [132…

```
%%sql
create table if not exists emails(
    ename text,
    email text
);
insert into emails values
('A','fdc@email.com'),
('C','fdoos@email.com');
select * from emails;
```

```
* sqlite:///practice3.db
```

Out[132…

	ename	email
0	A	fdc@email.com
1	C	fdoos@email.com

Get the name before the @ sign in email id:

In [133…

```
%%sql
SELECT
    ename,
```

```
    email,
    substr(email, 1, instr(email, '@') - 1) AS email_name
FROM emails;
```

```
* sqlite:///practice3.db
```

Out[133…

	ename	email	email_name
0	A	fdc@email.com	fdc
1	C	fdoos@email.com	fdoos

Get the domain name after the @ sign in email id:

In [134…

```
%%sql
SELECT
    ename,
    email,
    substr(email, instr(email, '@') + 1,length(email)) AS domain_name
FROM emails;
```

```
* sqlite:///practice3.db
```

Out[134…

	ename	email	domain_name
0	A	fdc@email.com	email.com
1	C	fdoos@email.com	email.com

Interview Question

In [138…

```
%%sql
create table if not exists manager_salary(
    id integer primary key,
    name text,
    salary integer,
    manager_id integer
);
```

```
* sqlite:///practice3.db
```

Out[138… —

In [139…

```
%%sql
delete from manager_salary;

insert into manager_salary values
    (1,'Joe',70000,3),
    (2,'Henry',80000,4),
    (3,'Sam',60000,null);

select * from manager_salary;
```

```
* sqlite:///practice3.db
```

Out[139…

	id	name	salary	manager_id
0	1	Joe	70000	3.0
1	2	Henry	80000	4.0
2	3	Sam	60000	NaN

Find employees who earn more than managers:

In [140…
```
%%sql
select a.id,a.name,a.salary from manager_salary a inner join manager_sala
```

```
* sqlite:///practice3.db
```

Out[140…

	id	name	salary
0	1	Joe	70000

Interview Question:

In [141…
```
%%sql
create table if not exists salary_dev(
    name text,
    salary integer
);
insert into salary_dev values
('n1',2831),
('n2', 1988),
('n3', 944);
select * from salary_dev;
```

```
* sqlite:///practice3.db
```

Out[141…

	name	salary
0	n1	2831
1	n2	1988
2	n3	944

Write a query to find out the deviation from average salary for the employesss who are getting more than average salary

In [ ]:
```
%%sql
select * from salary_dev where salary > (select avg(salary) from salary_d
```

```
* sqlite:///practice3.db
```

Out[ ]:

	name	salary
0	n1	2831
1	n2	1988

In [150…
```
%%sql
select *,avg(salary) over () as avg_salary from salary_dev
```

```
* sqlite:///practice3.db
```

Out[150…

	name	salary	avg_salary
0	n1	2831	1921.0
1	n2	1988	1921.0
2	n3	944	1921.0

Final Output:

In [154…
```
%%sql
select * from (select *,salary-avg(salary) over () as dev_salary from sal
where a.salary > (select avg(salary) from salary_dev);
```

```
* sqlite:///practice3.db
```

Out[154…

	name	salary	dev_salary
0	n1	2831	910.0
1	n2	1988	67.0

CASE WHEN END Statements in SQL with update query:

In [155…
```
%%sql
create table if not exists end_table(
    name text,
    state text
);
insert into end_table values
('n1','AS'),
('n2','BR'),
('n3','GA'),
('n4','GJ'),
('n5','HR');
select * from end_table;
```

```
* sqlite:///practice3.db
```

Out[155…

	name	state
0	n1	AS
1	n2	BR
2	n3	GA
3	n4	GJ
4	n5	HR

make a column for full name of state:

In [156…
```
%%sql
select *,case when state="AS" then 'Assam'
when state="BR" then 'Bihar'
when state="GA" then 'Goa'
when state="GJ" then 'Gujarat'
when state="HR" then 'Haryana' end as state_fullform
from end_table;
```

```
* sqlite:///practice3.db
```

Out[156…

	name	state	state_fullform
0	n1	AS	Assam
1	n2	BR	Bihar
2	n3	GA	Goa
3	n4	GJ	Gujarat
4	n5	HR	Haryana

In [157…

```
%%sql
update end_table set state = case when state="AS" then 'Assam'
when state="BR" then 'Bihar'
when state="GA" then 'Goa'
when state="GJ" then 'Gujarat'
when state="HR" then 'Haryana' end;
select * from end_table;
```

```
* sqlite:///practice3.db
```

Out[157…

	name	state
0	n1	Assam
1	n2	Bihar
2	n3	Goa
3	n4	Gujarat
4	n5	Haryana

when more than one condition are satisfied for an input then first condition will be executed, so sequence matters here

Interview Question:

In [160…

```
%%sql
drop table if exists f_and_m_table;
```

```
* sqlite:///practice3.db
```

Out[160… —

In [161…

```
%%sql
create table if not exists f_and_m_table(
    id integer primary key,
    name text,
    gender text,
    salary integer
);
insert into f_and_m_table values
(1,'n1','M',5000),
(2,'n2','F',6000),
(3,'n3','M',7000),
(4,'n4','F',8000),
```

```
(5,'n5','M',9000);
select * from f_and_m_table;
```

```
* sqlite:///practice3.db
```

Out[161…

	id	name	gender	salary
0	1	n1	M	5000
1	2	n2	F	6000
2	3	n3	M	7000
3	4	n4	F	8000
4	5	n5	M	9000

Write an SQL query to swap all 'f' and 'm' values with a single update statement.

In [162…

```
%%sql
update f_and_m_table set gender = case when gender='M' then 'F'
when gender='F' then 'M' end;
select * from f_and_m_table;
```

```
* sqlite:///practice3.db
```

Out[162…

	id	name	gender	salary
0	1	n1	F	5000
1	2	n2	M	6000
2	3	n3	F	7000
3	4	n4	M	8000
4	5	n5	F	9000

Interview Question:

In [165…

```
%%sql
drop table if exists revenue_table;
```

```
* sqlite:///practice3.db
```

Out[165… —

In [166…

```
%%sql
create table if not exists revenue_table(
    id integer,
    revenue integer,
    month text
);
insert into revenue_table values
(1,1000,'Jan'),
(2,1500,'Jan'),
(3,2000,'Feb'),
(1,2500,'Feb'),
(1,3000,'Mar');
select * from revenue_table;
```

```
* sqlite:///practice3.db
```

Out[166…

	id	revenue	month
0	1	1000	Jan
1	2	1500	Jan
2	3	2000	Feb
3	1	2500	Feb
4	1	3000	Mar

In [168…

```
%%sql
select id,
max(case when month='Jan' then revenue end) as jan_r,
max(case when month='Feb' then revenue end) as feb_r,
max(case when month='Mar' then revenue end) as mar_r
from revenue_table group by id
```

```
* sqlite:///practice3.db
```

Out[168…

	id	jan_r	feb_r	mar_r
0	1	1000.0	2500.0	3000.0
1	2	1500.0	NaN	NaN
2	3	NaN	2000.0	NaN

Interview Question:

In [169…

```
%%sql
create table if not exists score(
    id integer primary key,
    student text,
    score integer
);
insert into score values
(1,'Jack',1700),
(2,'Alice',2010),
(3,'Mike',2200),
(4,'Scott',2100);
select * from score;
```

```
* sqlite:///practice3.db
```

Out[169…

	id	student	score
0	1	Jack	1700
1	2	Alice	2010
2	3	Mike	2200
3	4	Scott	2100

write a query to return the two students with the closest test scores with the score difference

In [178…

```
%%sql
select * from (select *,rank() over(order by diff) rnk from (select id,le
```

```
score,
lead(score) over(order by score) next_score,
lead(score) over(order by score)-score diff
from score) a where a.diff is not null) where rnk<3
```

```
* sqlite:///practice3.db
```

Out[178…

	id	next_id	score	next_score	diff	rnk
0	2	3.0	2010	2100	90	1
1	4	NaN	2100	2200	100	2

CASE WHEN END statements with Aggregates group by in SQL:

In [179…

```
%%sql
create table if not exists aggregation_table(
    sid integer,
    marks integer
);
insert into aggregation_table values
(1,72),
(2,16),
(3,69),
(4,43),
(5,23);
select * from aggregation_table;
```

```
* sqlite:///practice3.db
```

Out[179…

	sid	marks
0	1	72
1	2	16
2	3	69
3	4	43
4	5	23

output we want:

63-100 -- Excellent.

33-62 -- pass.

0-32 -- Fail.

In [181…

```
%%sql
select *, case when marks between 63 and 100 then 'Excellent'
when marks between 33 and 62 then 'Pass'
when marks between 0 and 32 then 'Fail' end as performance
from aggregation_table;
```

```
* sqlite:///practice3.db
```

Out[181…

	sid	marks	performance
0	1	72	Excellent
1	2	16	Fail
2	3	69	Excellent
3	4	43	Pass
4	5	23	Fail

In [180…

```
%%sql
select case when marks between 63 and 100 then 'Excellent'
when marks between 33 and 62 then 'Pass'
when marks between 0 and 32 then 'Fail' end as performance,
count(*) as count_students
from aggregation_table group by performance;
```

```
* sqlite:///practice3.db
```

Out[180…

	performance	count_students
0	Excellent	2
1	Fail	2
2	Pass	1

Interview Question:

In [185…

```
%%sql
drop table if exists dataset_table;
```

```
* sqlite:///practice3.db
```

Out[185… —

In [186…

```
%%sql
create table if not exists dataset_table (
    orderid int,
    stateid text,
    status varchar(20),
    amount int
);
```

```
* sqlite:///practice3.db
```

Out[186… —

In [187…

```
%%sql
insert into dataset_table (orderid, stateid, status, amount) values
(1, 's1', 'shipped', 250),
(2, 's2', 'delivered', 150),
(3, 's3', 'packed', 300),
(4, 's2', 'shipped', 400),
(5, 's1', 'shipped', 500);
select * from dataset_table;
```

```
* sqlite:///practice3.db
```

Out[187…

	orderid	stateid	status	amount
0	1	s1	shipped	250
1	2	s2	delivered	150
2	3	s3	packed	300
3	4	s2	shipped	400
4	5	s1	shipped	500

we need to find no of orders shipped,delivered and packed

In [188…

```
%%sql
select stateid,
count(case when status='shipped' then orderid end) as shipped_orders,
count(case when status='delivered' then orderid end) as delivered_orders,
count(case when status='packed' then orderid end) as packed_orders
 from dataset_table group by stateid
```

```
* sqlite:///practice3.db
```

Out[188…

	stateid	shipped_orders	delivered_orders	packed_orders
0	s1	2	0	0
1	s2	1	1	0
2	s3	0	0	1

In [ ]: