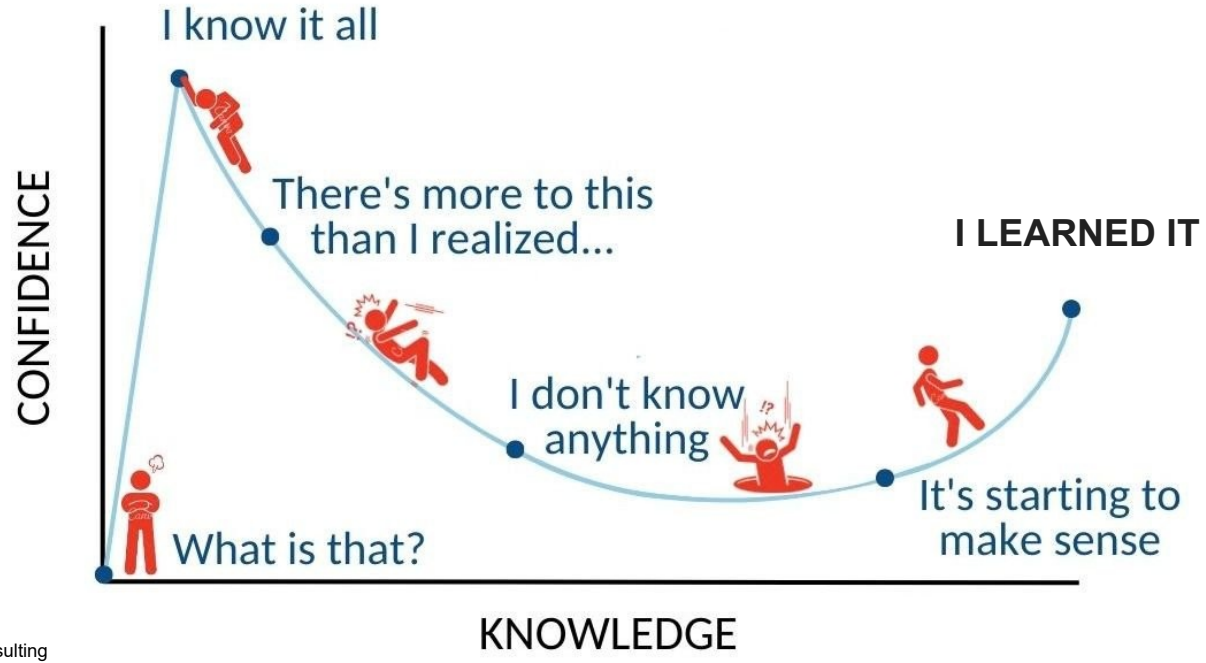
correlation.one

TECH FOR JOBS

Dunning-Kruger Effect



Credit: BVA Nudge Consulting

Query Structure

The SELECT clause can specify more than one column.

```
SELECT pet_type, pet_name
FROM people
WHERE pet_type = 'dog'
AND pet_age < 5;
```

Wildcard: % and _

The % will substitute zero, one, or multiple characters in a query.
In this example, all of the following are matches: Will, Willa, and Willows.

```
SELECT *
FROM actor
WHERE last_name LIKE 'Will%';
```

Wildcard: % and _

The _ will substitute only **one** character in a query.

_an returns all actors whose first name contains three letters, the second and third of which are an.

```
SELECT *
FROM actor
WHERE first_name LIKE '_an';
```

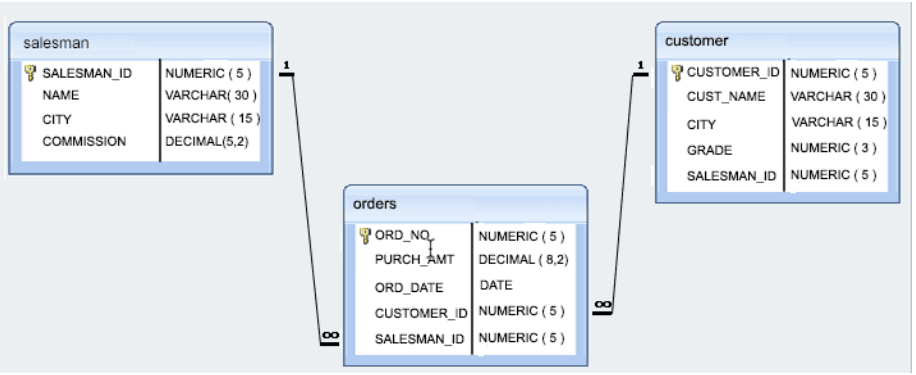
SELECT *

salesman_id	name	city	commission
5001	James Hoog	New York	0.15
5002	Nail Knite	Paris	0.13
5005	Pit Alex	London	0.11
5006	Mc Lyon	Paris	0.14
5003	Lauson Hense		0.12
5007	Paul Adam	Rome	0.13

SELECT * FROM salesman;

salesman_id	name	city	commission
5001	James Hoog	New York	0.15
5002	Nail Knite	Paris	0.13
5005	Pit Alex	London	0.11
5006	Mc Lyon	Paris	0.14
5003	Lauson Hense		0.12
5007	Paul Adam	Rome	0.13

SELECT COLUMNS IN DIFFERENT ORDER

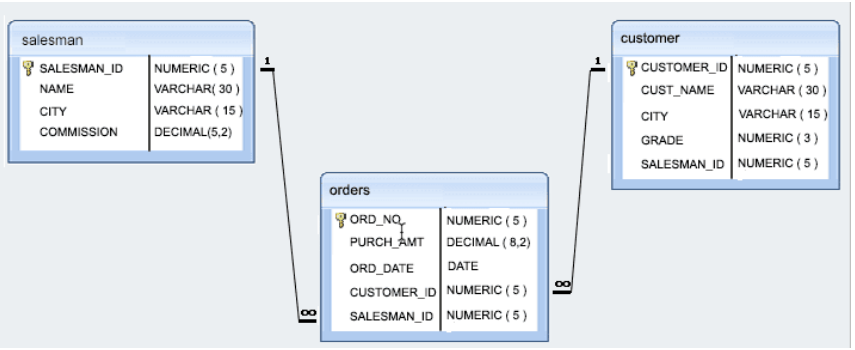


ord_no	purch_amt	ord_date	customer_id	salesman_id
70001	150.50	2012-10-05	3005	5002
70009	270.65	2012-09-10	3001	5005
70002	65.26	2012-10-05	3002	5001
70004	110.50	2012-08-17	3009	5003
70007	948.50	2012-09-10	3005	5002
70005	2400.60	2012-07-27	3007	5001
70008	5760.00	2012-09-10	3002	5001
70010	1983.43	2012-10-10	3004	5006
70003	2480.40	2012-10-10	3009	5003
70012	250.45	2012-06-27	3008	5002
70011	75.29	2012-08-17	3003	5007
70013	3045.60	2012-04-25	3002	5001

```
SELECT ord_date,salesman_id,ord_no,purch_amt
FROM  orders;
```

ord_date	salesman_id	ord_no	purch_amt
2012-10-05	5002	70001	150.50
2012-09-10	5005	70009	270.65
2012-10-05	5001	70002	65.26
2012-08-17	5003	70004	110.50
2012-09-10	5002	70007	948.50
2012-07-27	5001	70005	2400.60
2012-09-10	5001	70008	5760.00
2012-10-10	5006	70010	1983.43
2012-10-10	5003	70003	2480.40
2012-06-27	5002	70012	250.45

UNIQUE VALUES

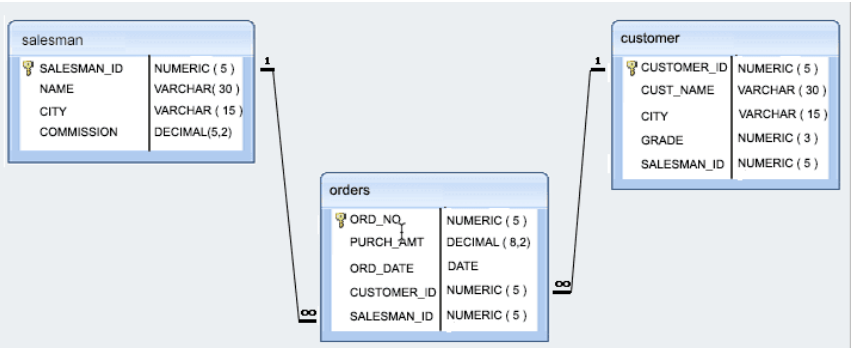


ord_no	purch_amt	ord_date	customer_id	salesman_id
70001	150.50	2012-10-05	3005	5002
70009	270.65	2012-09-10	3001	5005
70002	65.26	2012-10-05	3002	5001
70004	110.50	2012-08-17	3009	5003
70007	948.50	2012-09-10	3005	5002
70005	2400.60	2012-07-27	3007	5001
70008	5760.00	2012-09-10	3002	5001
70010	1983.43	2012-10-10	3004	5006
70003	2480.40	2012-10-10	3009	5003
70012	250.45	2012-06-27	3008	5002
70011	75.29	2012-08-17	3003	5007
70013	3045.60	2012-04-25	3002	5001

```
SELECT DISTINCT salesman_id
FROM  orders;
```

salesman_id
5006
5002
5001
5005
5003
5007

SPECIFY A CONDITION



customer_id	cust_name	city	grade	salesman_id
3002	Nick Rimando	New York	100	5001
3005	Graham Zusi	California	200	5002
3004	Fabian Johnson	Paris	300	5006
3007	Brad Davis	New York	200	5001
3009	Geoff Cameron	Berlin	100	5003
3008	Julian Green	London	300	5002
3003	Jozy Altidore	Moscow	200	5007
3001	Brad Guzan	London		5005

```
SELECT * FROM customer
WHERE grade =200;
```

customer_id	cust_name	city	grade	salesman_id
3002	Nick Rimando	New York	100	5001
3005	Graham Zusi	California	200	5002
3004	Fabian Johnson	Paris	300	5006
3007	Brad Davis	New York	200	5001
3009	Geoff Cameron	Berlin	100	5003
3008	Julian Green	London	300	5002
3003	Jozy Altidore	Moscow	200	5007
3001	Brad Guzan	London		5005

Aggregate functions allow you to perform a calculation on a set of values to return a single value.

Aggregate Functions

The most commonly used aggregate functions are:

AVG	Calculates the average of a set of values
COUNT	Counts the rows in a specific table or view
MIN	Returns the minimum value in a set of values
MAX	Returns the maximum value in a set of values
SUM	Calculates the sum of a set of values

Aggregate Functions

Aggregate functions are often used with:

01 The `GROUP BY` clause

02 The `HAVING` clause

03 The `SELECT` statement

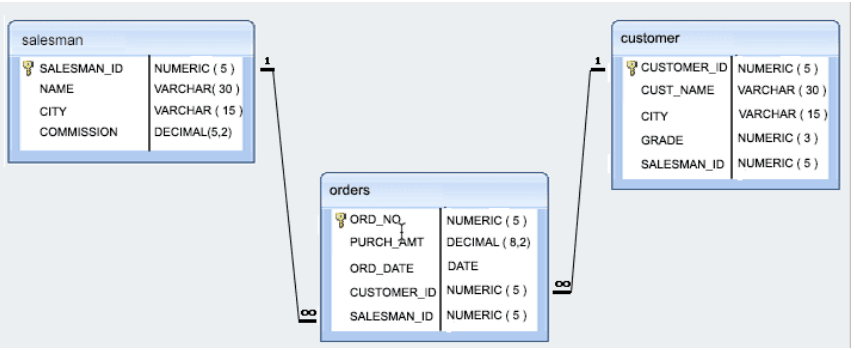
Aggregate Functions

Name	Value
A	10
A	20
B	40
C	20
C	50

```
SELECT
    Name,
    SUM(Value)
FROM
    sample_table
GROUP BY
    Name;
```

Name	SUM(Value)
A	30
B	40
C	70

Find the total purchase amount for all orders



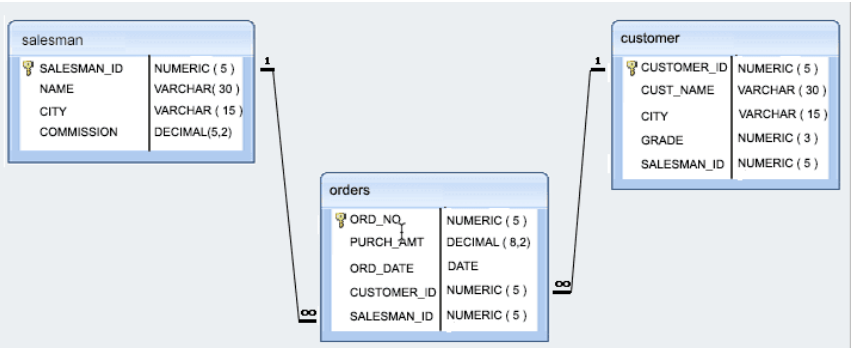
ord_no	purch_amt	ord_date	customer_id	salesman_id
70001	150.50	2012-10-05	3005	5002
70009	270.65	2012-09-10	3001	5005
70002	65.26	2012-10-05	3002	5001
70004	110.50	2012-08-17	3009	5003
70007	948.50	2012-09-10	3005	5002
70005	2400.60	2012-07-27	3007	5001
70008	5760.00	2012-09-10	3002	5001
70010	1983.43	2012-10-10	3004	5006
70003	2480.40	2012-10-10	3009	5003
70012	250.45	2012-06-27	3008	5002
70011	75.29	2012-08-17	3003	5007
70013	3045.60	2012-04-25	3002	5001

```
SELECT SUM (purch_amt)
FROM orders ;
```

purch_amt
150.50
270.65
65.26
110.50
948.50
2400.60
5760.00
1983.43
2480.40
250.45
75.29
3045.60

Sum : 17541.18

Highest purchase amount ordered by the each customer



ord_no	purch_amt	ord_date	customer_id	salesman_id
70001	150.50	2012-10-05	3005	5002
70009	270.65	2012-09-10	3001	5005
70002	65.26	2012-10-05	3002	5001
70004	110.50	2012-08-17	3009	5003
70007	948.50	2012-09-10	3005	5002
70005	2400.60	2012-07-27	3007	5001
70008	5760.00	2012-09-10	3002	5001
70010	1983.43	2012-10-10	3004	5006
70003	2480.40	2012-10-10	3009	5003
70012	250.45	2012-06-27	3008	5002
70011	75.29	2012-08-17	3003	5007
70013	3045.60	2012-04-25	3002	5001

```
SELECT customer_id, MAX(purch_amt)
FROM orders
GROUP BY customer_id ;
```

customer_id	purch_amt
3005	150.5
3001	270.65
3002	65.26
3009	110.5
3005	948.5
3007	2400.6
3002	5760
3004	1983.43
3009	2480.4
3008	250.45
3003	75.29
3002	3045.6

customer_id	max
3004	1983.43
3008	250.45
3001	270.65
3007	2400.6
3005	948.5
3002	5760
3009	2480.4

Order By Aggregates

The `ORDER BY` function:

- Is added towards the end of a query.
- Returns in an ascending order by default.
- Can also return in a descending order by adding `DESC`.
- Can limit the return by adding `LIMIT`.
- **NOTE:** Use the `ROUND` function to round up the number after the the decimal.

SQL Query

```sql
SELECT *
FROM individual
JOIN drivers ON individual.driver_id = drivers.id
WHERE individual.id = 647 OR individual.id=146 OR individual.id=981 OR individual.id=45;
```

SQL Query

Name = results

```
SELECT *
FROM individual
JOIN drivers ON individual.driver_id = drivers.id
WHERE individual.id = 647 OR individual.id=146 OR individual.id=981 OR individual.id=45;
```

```
SELECT * FROM results;
```

SQL Query - Subquery

```sql
SELECT *
FROM individual
JOIN drivers ON individual.driver_id = drivers.id
WHERE individual.id = 647 OR individual.id=146 OR individual.id=981 OR individual.id=45;
```

```sql
SELECT * FROM (
        SELECT *
            FROM individual
            JOIN drivers ON individual.driver_id = drivers.id
            WHERE individual.id = 647 OR individual.id=146 OR individual.id=981 OR individual.id=45
           );
```

Subqueries

A subquery is nested inside a larger query. Subqueries occur in:

01 The SELECT statement

02 The FROM clause

03 The WHERE clause

SQL Query - CTE

```sql
SELECT *
FROM individual
JOIN drivers ON individual.driver_id = drivers.id
WHERE individual.id = 647 OR individual.id=146 OR individual.id=981 OR individual.id=45;
```

```sql
WITH cte_suspects AS
(
    SELECT *
    FROM individual
    JOIN drivers ON individual.driver_id = drivers.id
    WHERE individual.id = 647 OR individual.id=146 OR individual.id=981 OR individual.id=45
)

SELECT * FROM cte_suspects;
```

SQL Query - View

```sql
SELECT *
FROM individual
JOIN drivers ON individual.driver_id = drivers.id
WHERE individual.id = 647 OR individual.id=146 OR ind
```

```sql
dual.id=45;
```

- crime_database.db
  - Tables
    - crime_scene
    - drivers
    - facebook_event
    - gym_affiliated
    - gym_record
    - individual
    - interrogation
  - Views
    - v_suspects

```sql
CREATE VIEW IF NOT EXISTS v_suspects AS
    SELECT *
    FROM individual JOIN drivers ON individual.driver_id = drivers.id
    WHERE individual.id = 647 OR individual.id=146 OR individual.id=981 OR individual.id=45;
```

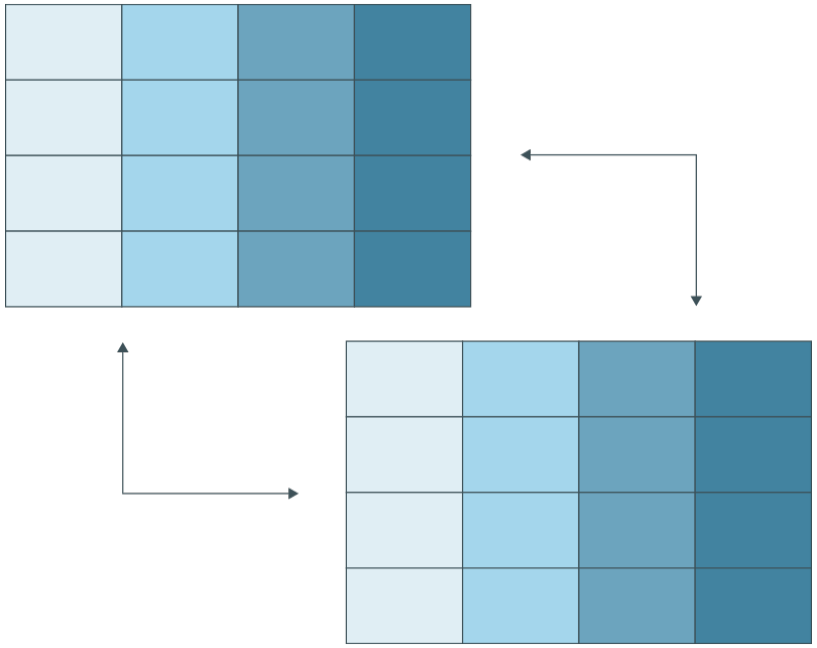
```sql
SELECT * FROM v_suspects;
```

A **view** is a virtual table that can be created from a single table, multiple tables, or another view.

SQL Views

Views are created by using the `CREATE VIEW` statement.

Views are created from a single table, multiple tables, or another view.



SQL Order of Execution

ORDER	CLAUSE	FUNCTION
1	from	Choose and join tables to get base data.
2	where	Filters the base data.
3	group by	Aggregates the base data.
4	having	Filters the aggregated data.
5	select	Returns the final data.
6	order by	Sorts the final data.
7	limit	Limits the returned data to a row count.