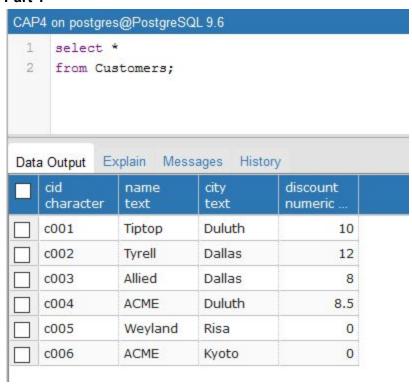
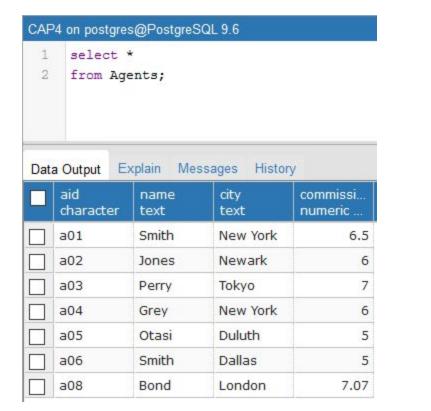
Part 1





# CAP4 on postgres@PostgreSQL 9.6

select \*

2 from Products;

Dat	Data Output Explain Messages History										
	pid character	name text	city text	quantity integer	priceusd numeric						
	p01	comb	Dallas	111400	0.5						
	p02	brush	Newark	203000	0.5						
	p03	razor	Duluth	150600	1						
	p04	pen	Duluth	125300	1						
	p05	pencil	Dallas	221400	1						
	p06	trapper	Dallas	123100	2						
	p07	case	Newark	100500	1						
	p08	eraser	Newark	200600	1.25						

# CAP4 on postgres@PostgreSQL 9.6

- 1 select \*
- 2 from Orders;

Data Output Explain Messages History										
	ordnumb integer	month character	cid character	aid character	pid character	qty integer	totalusd numeric			
	1011	Jan	c001	a01	p01	1000	450			
	1012	Jan	c002	a03	p03	1000	880			
	1015	Jan	c003	a03	p05	1200	1104			
	1016	Jan	c006	a01	p01	1000	500			
	1017	Feb	c001	a06	p03	600	540			
	1018	Feb	c001	a03	p04	600	540			
	1019	Feb	c001	a02	p02	400	180			
	1020	Feb	c006	a03	p07	600	600			
	1021	Feb	c004	a06	p01	1000	460			
	1022	Mar	c001	a05	p06	400	720			
	1023	Mar	c001	a04	p05	500	450			
	1024	Mar	c006	a06	p01	800	400			
	1025	Apr	c001	a05	p07	800	720			
	1026	May	c002	a05	p03	800	744			

#### Part 2

A **superkey** is a column or set of columns that can be used to uniquely identify each row based on their values. In essence, no row will have the same superkey value as another.

A candidate key is a superkey that uses the fewest number of columns to uniquely identify each row. It's the simplest superkey. For example, while <code>employeeID</code>, <code>name</code>, and <code>gender</code> may be a superkey, <code>employeeID</code> would likely be a candidate key.

A **primary key** is the candidate key that is chosen to be used to uniquely identify each row. This is done to simplify things and avoid confusingly referencing the same entry in different ways.

## Part 3

Let's assume we have a system recording conference room reservations, and we log these reservations in a table. This table would be called Reservations. In it, we would have: employeeID (int (foreign key), non-nullable), timeCreated(timestamp, non-nullable), meeting description (text, nullable), meetingStart (time, non-nullable), meetingEnd (time, non-nullable), roomID (int (foreignkey), non-nullable).

## Part 4

First Normal Form Rule - Each attribute should not be able to be further divided or be constituted of multiple values. For example, instead of name: 'Bob G. Goodman', you would want firstName: 'Bob', middleInitial: 'G', lastName 'Goodman'. This is important because it keeps information in a consistent and usable format. Plus, when it is combined, you cannot easily break it down and access the parts, such as the last name.

**Access Rows by content only -** When looking for rows, you do not access them by position, but rather the content within them. That is to say, do not ask for row 5, but instead the rows with the value of \$25.00. This is because there is no set order for your rows; the way they are returned may not be the same next time.

**All rows must be unique -** Simply put, you should never have two or more rows that are exactly the same. For one, since we are technically working with sets, having duplicates is not allowed according to set theory. Second, having two entries being the exact same means there is no way to distinguish between the two, and you cannot get only one of them.