Session 15 & 16

Implementing KPIs (SQL IV)

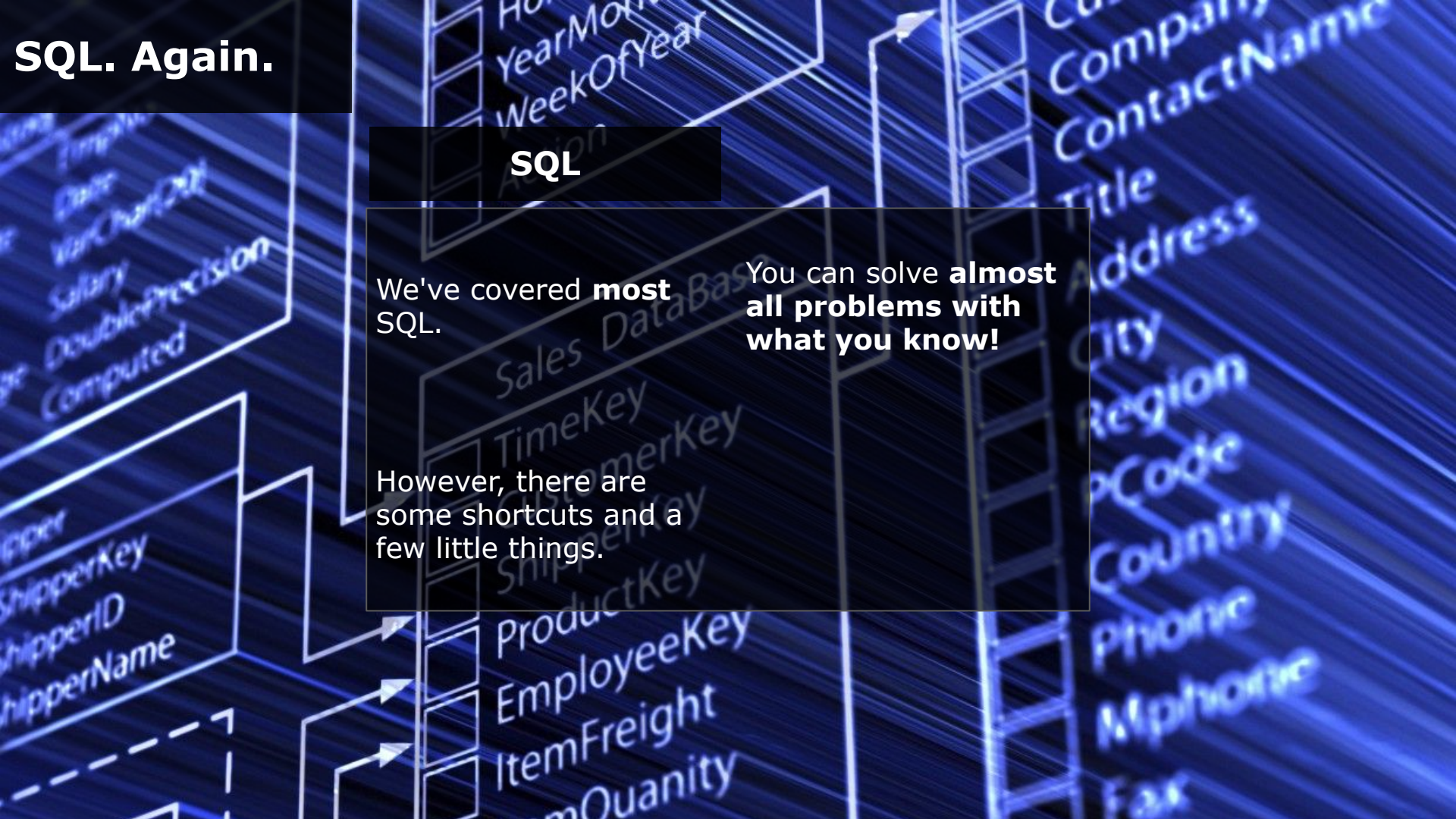
SQL. Again.

SQL

We've covered **most** SQL.

However, there are some shortcuts and a few little things.

You can solve **almost all problems with what you know!**



SQL. Again.

SQL

Thinking in SQL takes practice.

The more examples you see the better!

```
Today...
Example 1
Example 2
Example 3
Example 4
Example 5
Example 6
Example 7
```

Through these examples we'll also introduce a few new SQL concepts and syntax.

SQL. Again.

SQL & KPIs

SQL in practice is your key "go to" as an analyst for converting **KPI's** to numbers for your business.

SQL. Again.

SQL & KPIs

SQL in practice is your key "go to" as an analyst for converting **KPI's** to numbers for your business.

KPI: Key performance indicator.

A **measurable value** that demonstrates how effectively a company is achieving key business objectives.

SQL. Again.

SQL & KPIs

SQL in practice is your key "go to" as an analyst for converting **KPI's** to numbers for your business.

KPI: Key performance indicator.

A **measurable value** that demonstrates how effectively a company is achieving key business objectives.

Determining what to measure. May or may not be your job.

Determining how to measure it. Likely your job.

Actually getting the data. Almost certainly your job.

"Standard" KPI's may have a specific formula.

Or they may be something your boss wants **you** to measure. You get to **formalize how it is measured & measure it.**

SQL. Again.

SQL & KPIs

SQL in practice is your key "go to" as an analyst for converting **KPI's** to numbers for your business.

KPI: Key performance indicator.

A **measurable value** that demonstrates how effectively a company is achieving key business objectives.

Determining what to measure.
May or may not be your job.

Determining how to measure it.
Likely your job.

Actually getting the data.
Almost certainly your job.

"Standard" KPI's may have a specific formula.

Or they may be something your boss wants **you** to measure. You get to **formalize how it is measured & measure it.**

Examples of KPIs:

Retail
Sales & Gross Margin

Sales per Square Foot

Average customer spend

Stock turnover rate

Return on investment on marketing spend

Growth

Engagement

Retention rate

Churn rate

Supply chain
Back order rate

Rate of return

% of out-of-stock items

Perfect order rate

Inventory turnover

Carrying cost of inventory

Inventory Accuracy

SQL. Again.

SQL & KPIs

SQL in practice is your key "go to" as an analyst for converting **KPI's** to numbers for your business.

KPI: Key performance indicator.

A **measurable value** that demonstrates how effectively a company is achieving key business objectives.

Determining what to measure.
May or may not be your job.

Determining how to measure it.
Likely your job.

Actually getting the data.
Almost certainly your job.

"Standard" KPI's may have a specific formula.

Or they may be something your boss wants **you** to measure. You get to **formalize how it is measured & measure it.**

Examples of KPIs:

Retail
Sales & Gross Margin

Sales per Square Foot

Average customer spend

Stock turnover rate

Return on investment on marketing spend

Growth

Engagement

Retention rate

Churn rate

Supply chain
Back order rate

Rate of return

% of out-of-stock items

Perfect order rate

Inventory turnover

Carrying cost of inventory

Inventory Accuracy

Example 1

Growth
Engagement
Retention rate

Step 1:
Write down a concrete measurable description of the KPI

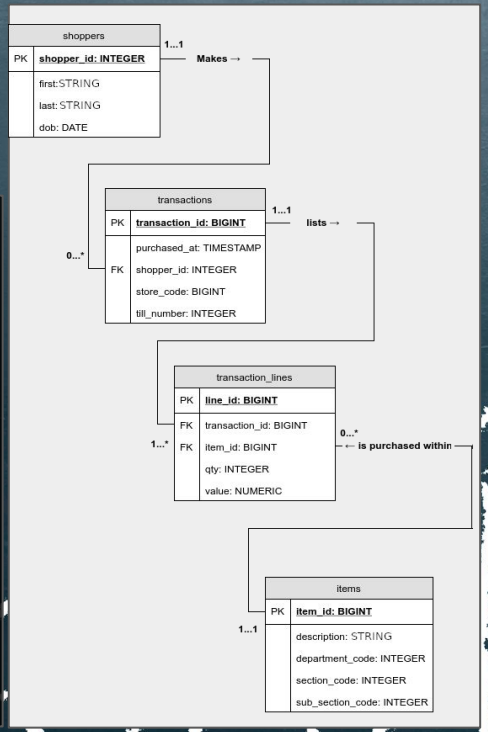
Step 2. Write in SQL.

Example 1

Growth
Engagement
Retention rate

Step 1:
Write down a concrete measurable description of the KPI

Growth, measured as
→ Count of active customers per month (KPI 1)



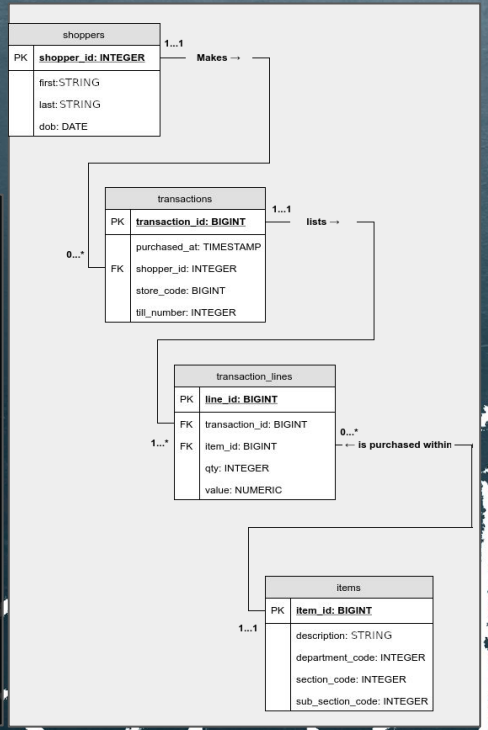
Step 2. Write in SQL.

Example 1

Growth
Engagement
Retention rate

Step 1:
Write down a concrete measurable description of the KPI

Growth, measured as
→ Count of active customers per month (KPI 1)



Step 2. Write in SQL.

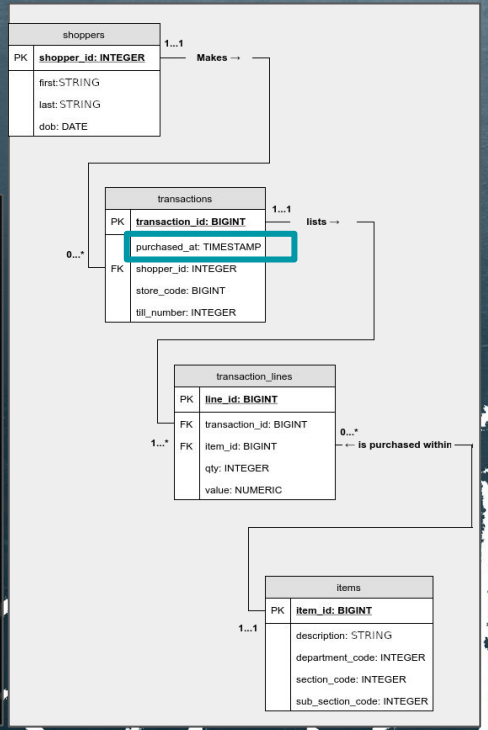
```
SELECT
FROM
```

Example 1

Growth
Engagement
Retention rate

Step 1:
Write down a concrete measurable description of the KPI

Growth, measured as
→ Count of active customers per month (KPI 1)



Step 2. Write in SQL.

```
SELECT
FROM transactions
GROUP BY DATE_TRUNC('month',purchased_at)
```

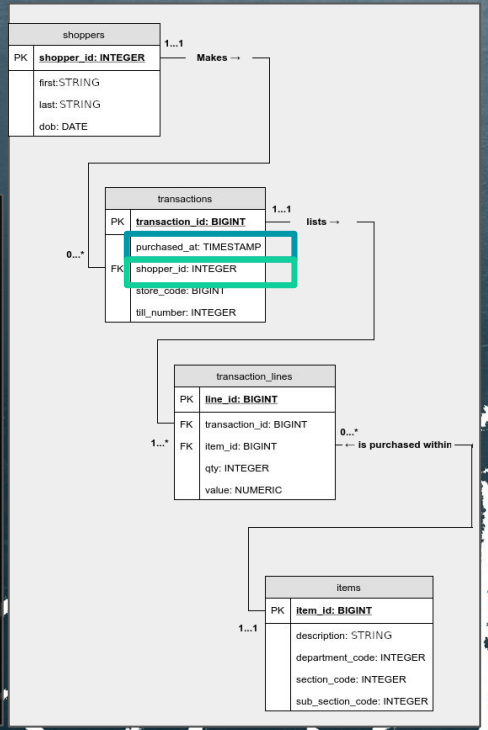
Example 1

Growth
Engagement
Retention rate

Step 1:
Write down a concrete measurable description of the KPI

Growth, measured as
→ Count of active customers per month (KPI 1)

Implicitly we mean distinct here, otherwise we'd be counting customer visits



Step 2. Write in SQL.

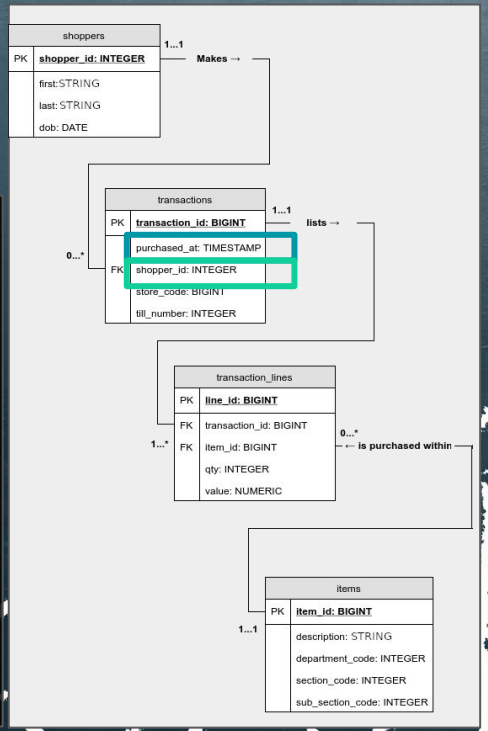
```
SELECT COUNT( DISTINCT shopper_id )
FROM transactions
GROUP BY DATE_TRUNC('month', purchased_at)
```

Example 1

Growth
Engagement
Retention rate

Step 1:
Write down a concrete measurable description of the KPI

Growth, measured as
→ Count of active customers per month (KPI 1)



Step 2. Write in SQL.

```
SELECT COUNT( DISTINCT shopper_id ) AS active_customers,
 DATE_FORMAT(DATE_TRUNC('month',purchased_at),'yyyy-MM') AS mth
FROM transactions
GROUP BY DATE_TRUNC('month',purchased_at)
ORDER BY 2 DESC;
```

We now "pretty-up" the output so:
(1) it gives the headings we want

and

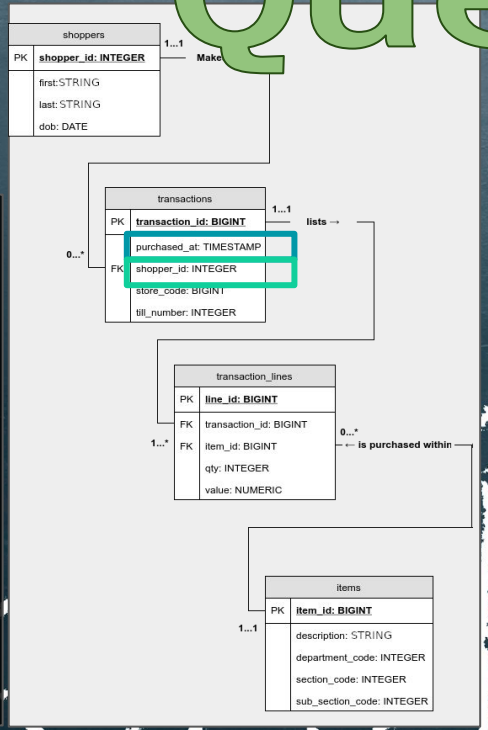
(2) it presents to the user in the order we want.

Questions?

Growth
Engagement
Retention rate

Step 1:
Write down a concrete measurable description of the KPI

Growth, measured as
→ Count of active customers per month (KPI 1)



Step 2. Write in SQL.

```
SELECT COUNT( DISTINCT shopper_id ) AS active_customers,
 DATE_FORMAT(DATE_TRUNC('month',purchased_at),'yyyy-MM') AS mth
FROM transactions
GROUP BY DATE_TRUNC('month',purchased_at)
ORDER BY 2 DESC;
```

We now "pretty-up" the output so:
(1) it gives the headings we want

and

(2) it presents to the user in the order we want.

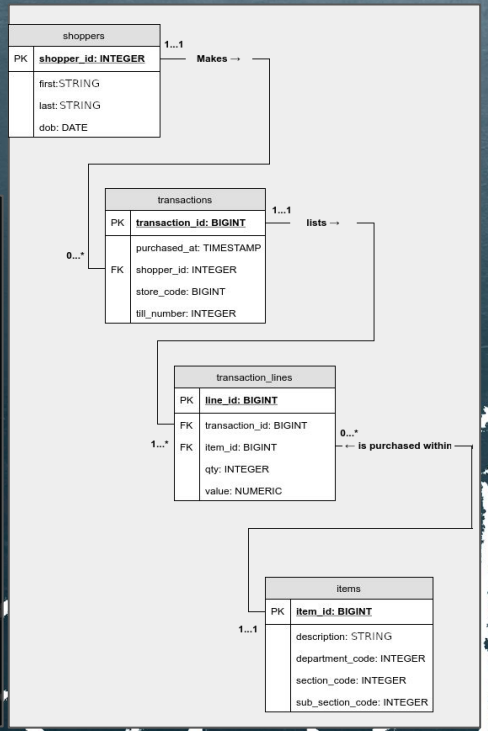
Example 2

Growth
Engagement
Retention rate

Step 1:
Write down a concrete measurable description of the KPI

Growth, measured as
→ Count of active customers per month (KPI 1)

→ Total sales per month for store 5000128068369 (KPI 2)



Step 2. Write in SQL.

Your turn 1

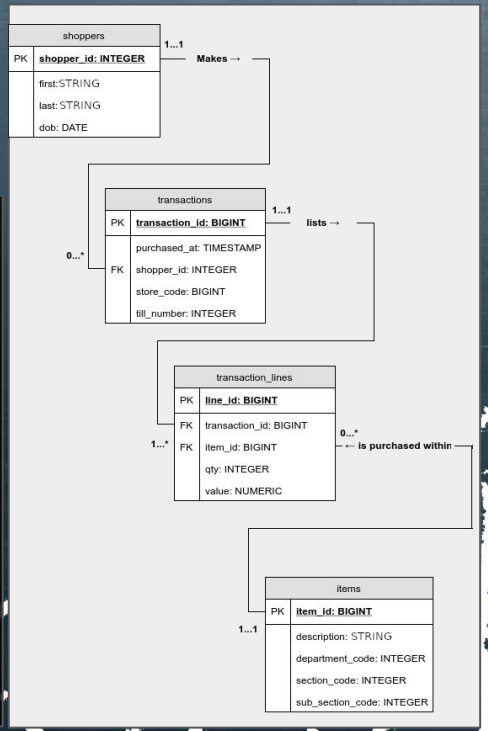
Example 2

Growth
Engagement
Retention rate

Step 1:
Write down a concrete measurable description of the KPI

Growth, measured as
→ Count of active customers per month (KPI 1)

→ Total sales per month for store 5000128068369 (KPI 2)



Step 2. Write in SQL.

```sql
SELECT SUM(value) AS total_sales,
 DATE_FORMAT(DATE_TRUNC('month',purchased_at),'yyyy-MM') AS mth
FROM transactions
JOIN transaction_lines
USING (transaction_id)
WHERE store_code = 5000128068369
GROUP BY DATE_TRUNC('month',purchased_at);
```

Example 3

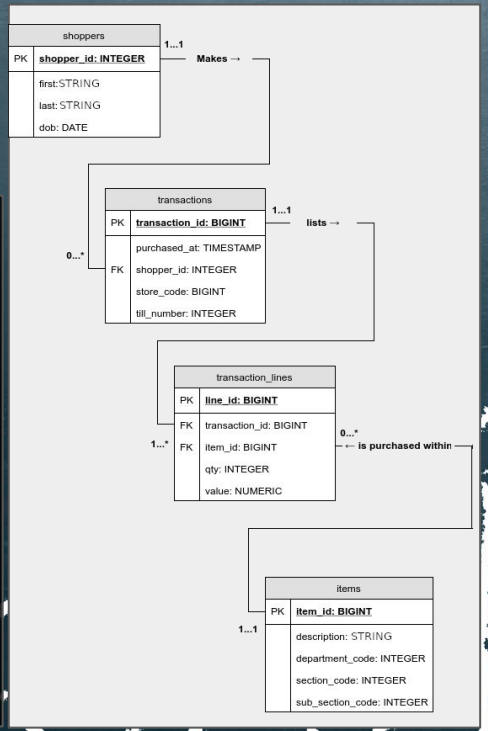
Growth
Engagement
Retention rate

Step 1:
Write down a concrete measurable description of the KPI

Growth, measured as
→ Count of active customers per month (KPI 1)

→ Total sales per month for store 5000128068369 (KPI 2)

→ Total sales per month, per store (KPI 3)



Step 2. Write in SQL.

```sql
SELECT store_code, SUM(value) AS total_sales,
 DATE_FORMAT(DATE_TRUNC('month',purchased_at),'yyyy-MM') AS mth
FROM transactions
JOIN transaction_lines
USING (transaction_id)
WHERE store_code = 5000128068369
GROUP BY DATE_TRUNC('month',purchased_at), store_code
```

Example 3

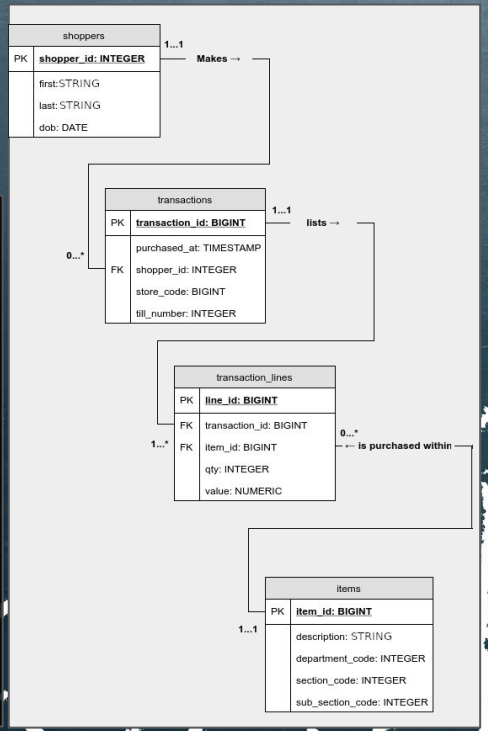
Growth
Engagement
Retention rate

Step 1:
Write down a concrete measurable description of the KPI

Growth, measured as
→ Count of active customers per month (KPI 1)

→ Total sales per month for store 5000128068369 (KPI 2)

→ Total sales per month, per store (KPI 3)



Step 2. Write in SQL.

```sql
SELECT store_code, SUM(value) AS total_sales,
 DATE_FORMAT(DATE_TRUNC('month',purchased_at),'yyyy-MM') AS mth
FROM transactions
JOIN transaction_lines
USING (transaction_id)
GROUP BY DATE_TRUNC('month',purchased_at), store_code;
```

But what about "prettying up"?
Two options.

→ Use order by.
→ Graph / visualize

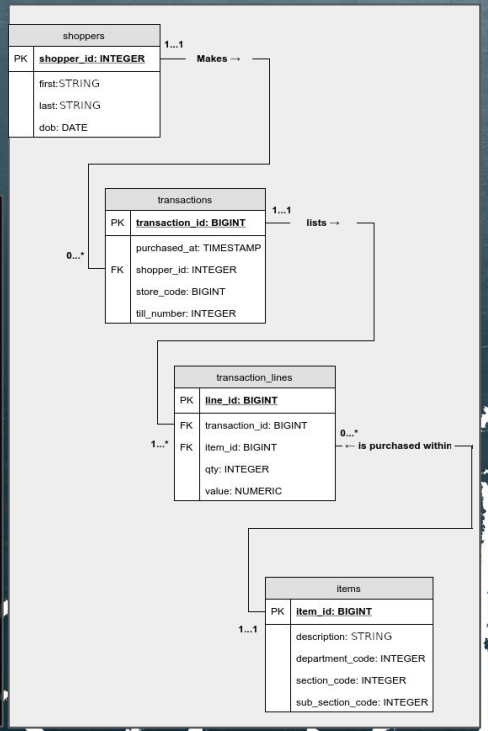
Demo!

Example 4

Growth
Engagement
Retention rate

Step 1:
Write down a concrete measurable description of the KPI

Engagement, measured as
→ Count of repeat customers per month for store 5000128068369 (KPI 4)



Step 2. Write in SQL.

```
SELECT
FROM transactions
WHERE store_code = 5000128068369
GROUP BY DATE_TRUNC('month', purchased_at)
```

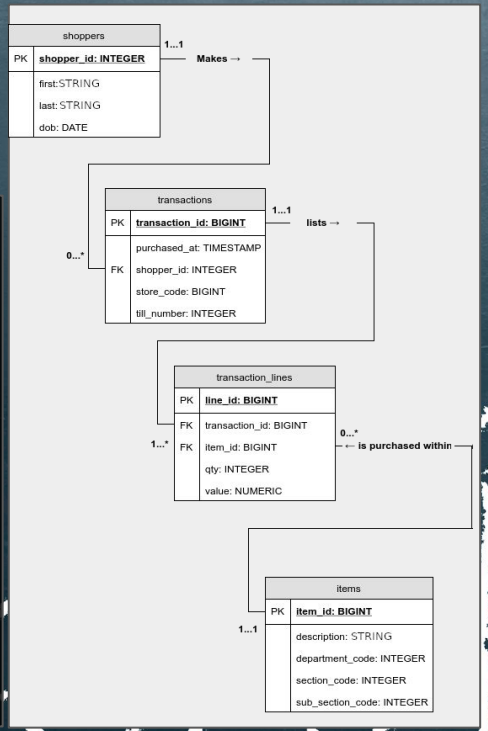
Question: How to encode the notion of "repeat"?

Example 4

Growth
Engagement
Retention rate

Step 1:
Write down a concrete measurable description of the KPI

Engagement, measured as
→ Count of repeat customers per month for store 5000128068369 (KPI 4)



Step 2. Write in SQL.

```
SELECT
FROM transactions
WHERE store_code = 5000128068369
GROUP BY DATE_TRUNC('month', purchased_at)
```

Question: How to encode the notion of "repeat"?

Answer: In each bucket, only count shoppers if they have two different transaction_ids in the bucket.

→ What assumptions have I made?

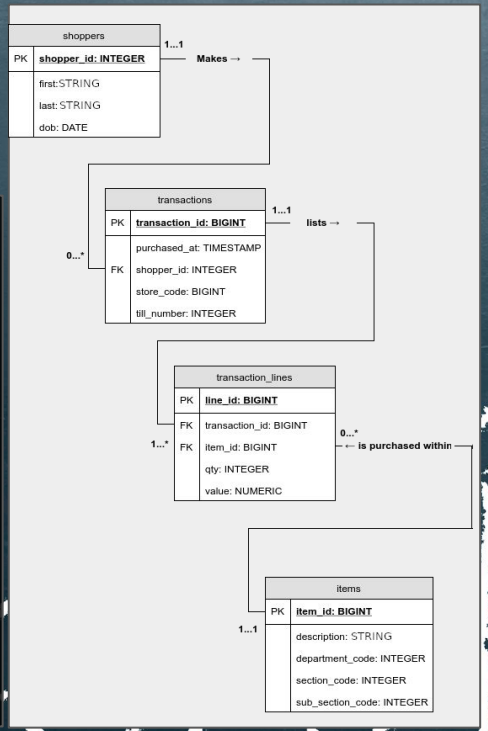
Example 4

Growth
Engagement
Retention rate

Step 1:
Write down a concrete measurable description of the KPI

Engagement, measured as
→ Count of repeat customers per month for store 5000128068369 (KPI 4)

KPI 4 Assumptions:
→ Returning on the same day still counts as a repeat customer.



Step 2. Write in SQL.

```
SELECT
FROM transactions
WHERE store_code = 5000128068369
GROUP BY DATE_TRUNC('month', purchased_at)
```

Question: How to encode the notion of "repeat"?

Answer: In each bucket, only count shoppers if they have two different transaction_ids in the bucket.

→ What assumptions have I made?

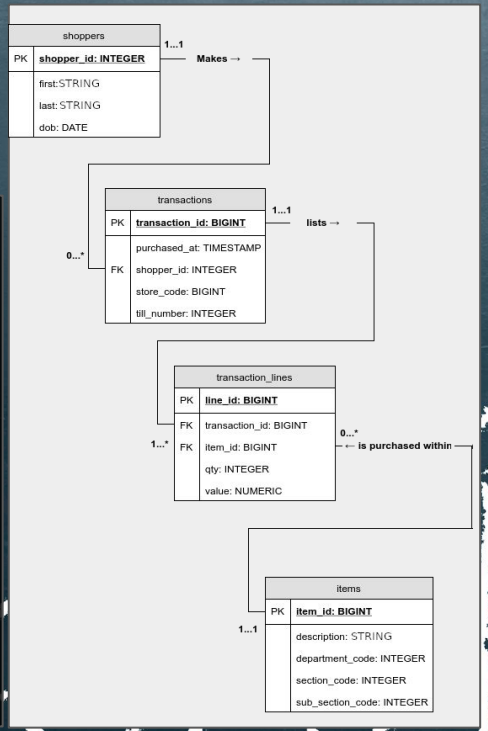
Example 4

Growth
Engagement
Retention rate

Step 1:
Write down a concrete measurable description of the KPI

Engagement, measured as
→ Count of repeat customers per month for store 5000128068369 (KPI 4)

KPI 4 Assumptions:
→ Returning on the same day still counts as a repeat customer.



Step 2. Write in SQL.

```
SELECT DATE_TRUNC('month', purchased_at) as mth, shopper_id
FROM transactions
WHERE store_code = 5000128068369
GROUP BY DATE_TRUNC('month', purchased_at), shopper_id
```

Counting buckets (groups) takes two steps.
→ One query to make a table with one row per bucket [the subquery].
→ One query to count the rows

The subquery:
First group by month + shopper_id.

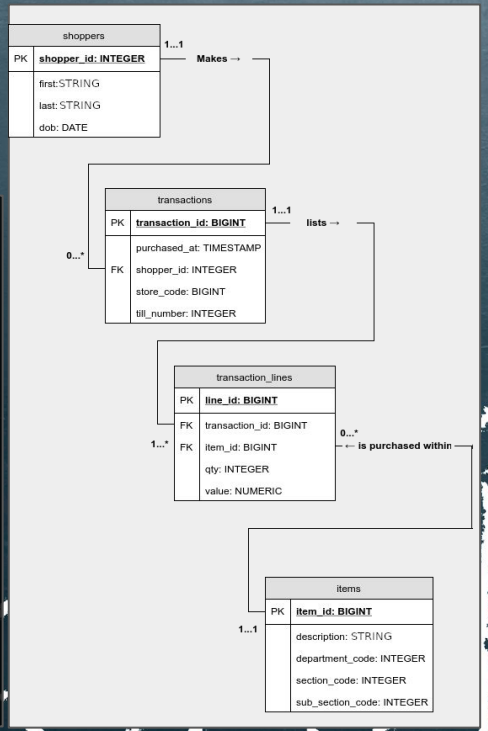
Example 4

Growth
Engagement
Retention rate

Step 1:
Write down a concrete measurable description of the KPI

Engagement, measured as
→ Count of repeat customers per month for store 5000128068369 (KPI 4)

KPI 4 Assumptions:
→ Returning on the same day still counts as a repeat customer.



Step 2. Write in SQL.

```
SELECT DATE_TRUNC('month', purchased_at) as mth, shopper_id
FROM transactions
WHERE store_code = 5000128068369
GROUP BY DATE_TRUNC('month', purchased_at), shopper_id
HAVING COUNT(DISTINCT transaction_id) > 1
```

First group by month + shopper_id.

Keep only those rows where the count of distinct transaction_ids > 1.

Question: What does this table represent?

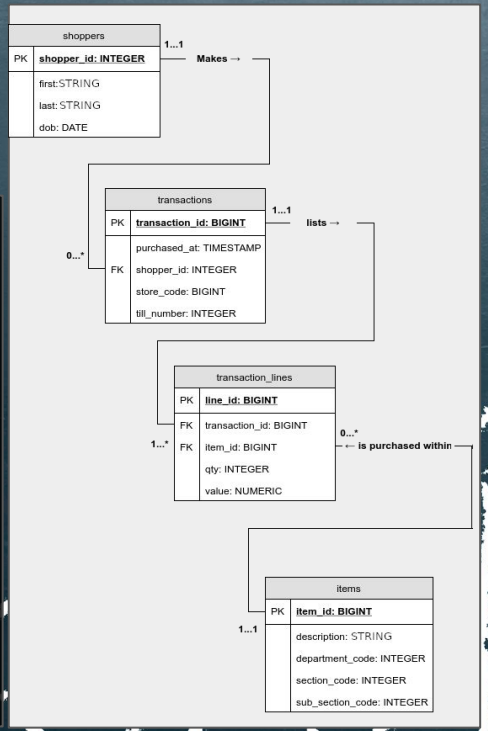
Example 4

Growth
Engagement
Retention rate

Step 1:
Write down a concrete measurable description of the KPI

Engagement, measured as
→ Count of repeat customers per month for store 5000128068369 (KPI 4)

KPI 4 Assumptions:
→ Returning on the same day still counts as a repeat customer.



Step 2. Write in SQL.

```
SELECT DATE_TRUNC('month', purchased_at) as mth, shopper_id
FROM transactions
WHERE store_code = 5000128068369
GROUP BY DATE_TRUNC('month', purchased_at), shopper_id
HAVING COUNT(DISTINCT transaction_id) > 1
```

First group by month + shopper_id.

Keep only those rows where the count of distinct transaction_ids > 1.

Question: What does this table represent?

Answer: Pairs of shopper_ids and months, where the shopper shopped at least twice in the month.

Fact known per row:
shopper_id shopped at least twice in mth

Example 4

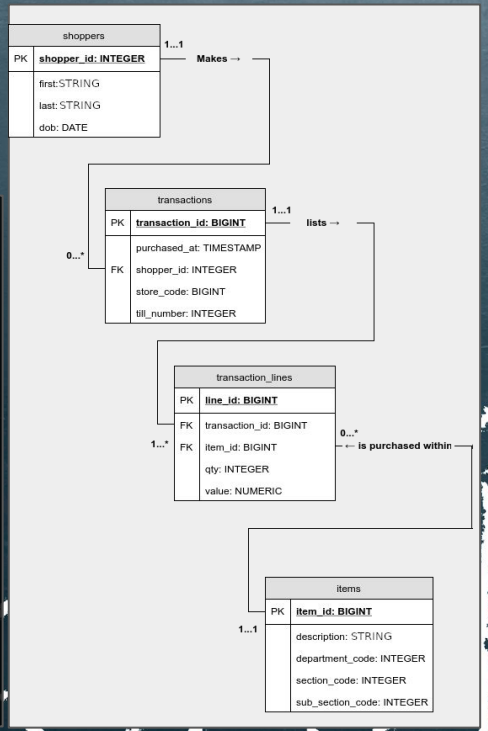
Growth
Engagement
Retention rate

Step 1:

Write down a concrete measurable description of the KPI

Engagement, measured as
→ Count of repeat customers per month for store 5000128068369 (KPI 4)

KPI 4 Assumptions:
→ Returning on the same day still counts as a repeat customer.



Step 2. Write in SQL.

```
SELECT date_format(mth, 'yyyy-MM') AS mth, COUNT(*)
FROM (
        SELECT DATE_TRUNC('month', purchased_at) as mth,
                shopper_id
        FROM transactions
        WHERE store_code = 5000128068369
        GROUP BY DATE_TRUNC('month', purchased_at), shopper_id
        HAVING COUNT(DISTINCT transaction_id) > 1
) x
GROUP BY mth;
```

Now use a subquery to group into and count.

Question: What does this table represent?

Answer: Pairs of shopper_ids and months, where the shopper shopped twice in the month.

Fact known per row:
shopper_id shopped twice in mth

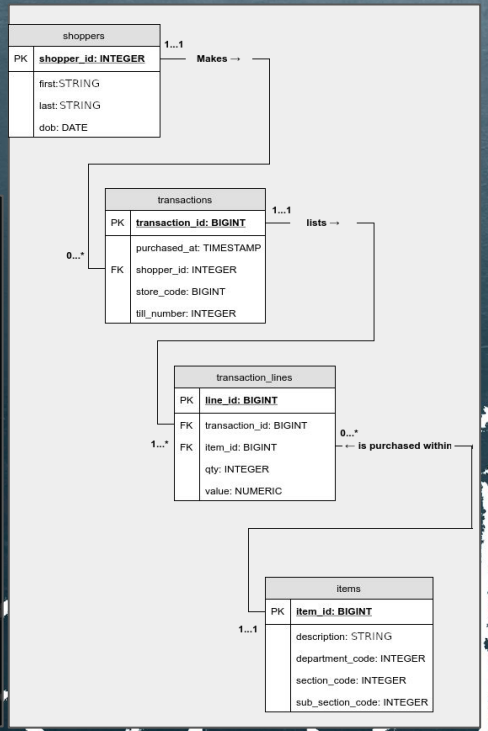
Example 5

Growth
Engagement
Retention rate

Step 1:
Write down a concrete measurable description of the KPI

Engagement, measured as
→ Count of repeat customers per month for store 5000128068369 (KPI 4)

KPI 4 Assumptions:
→ Returning is defined as returning on a separate day.



Step 2. Write in SQL.

```
SELECT date_format(mth, 'yyyy-MM') AS mth, COUNT(*)
FROM (
        SELECT DATE_TRUNC('month', purchased_at) as mth,
               shopper_id
        FROM transactions
        WHERE store_code = 5000128068369
        GROUP BY DATE_TRUNC('month', purchased_at), shopper_id
        HAVING COUNT(DISTINCT transaction_id) > 1
) x
GROUP BY mth;
```

Code from Example 4

Your turn 2

Example 5

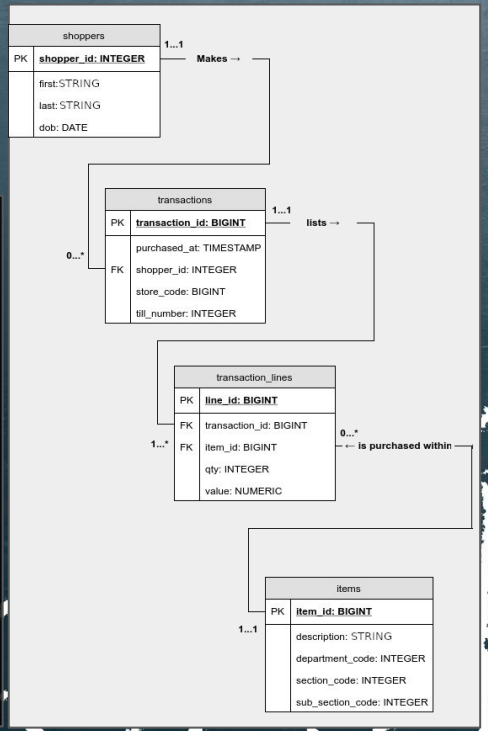
Growth
Engagement
Retention rate

Step 1:

Write down a concrete measurable description of the KPI

Engagement, measured as
→ Count of repeat customers per month for store 5000128068369 (KPI 5)

KPI 5 Assumptions:
→ Returning is defined as returning on a separate day.



Step 2. Write in SQL.

```
SELECT date_format(mth, 'yyyy-MM') AS mth, COUNT(*)
FROM (
        SELECT DATE_TRUNC('month', purchased_at) as mth,
               shopper_id
        FROM transactions
        WHERE store_code = 5000128068369
        GROUP BY DATE_TRUNC('month', purchased_at), shopper_id
        HAVING COUNT(DISTINCT purchased_at::DATE) > 1
) x
GROUP BY mth;
```

Q. How to encode the notion of "repeat on different days"

A1. Count of distinct purchased_at date (as a DATE) > 1

A2.
MAX(purchased_at::DATE) - MIN(purchased_at::DATE) > 0

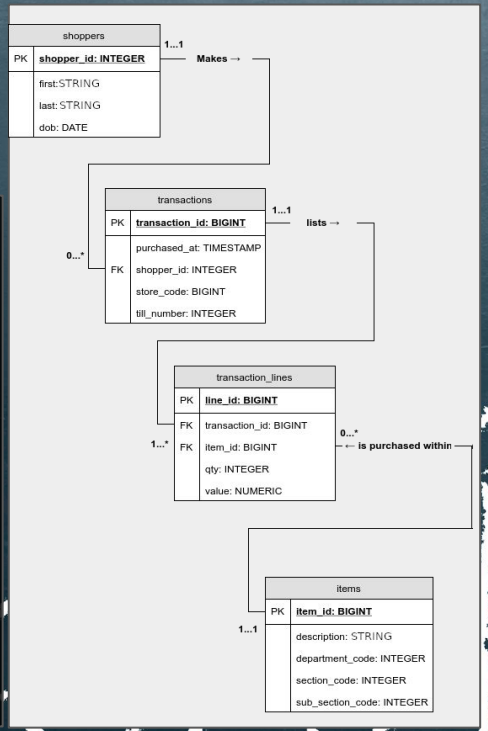
Example 6

Growth
Engagement
Retention rate

Step 1:
Write down a concrete measurable description of the KPI

Engagement, measured as
→ Count of new customers per month
(KPI 6)

KPI 6 Assumptions:
→ Customers are considered new if they have never been seen before.



Step 2. Write in SQL.

Your turn 3

Example 6

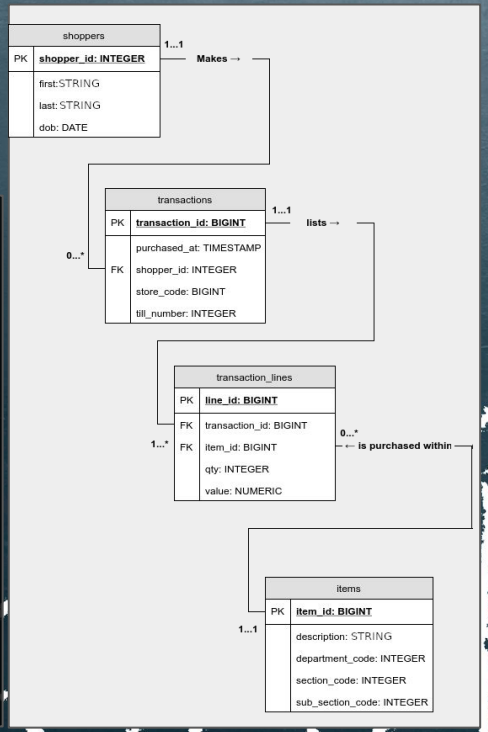
Growth
Engagement
Retention rate

Step 1:

Write down a concrete measurable description of the KPI

Engagement, measured as
→ Count of new customers per month
(KPI 6)

KPI 6 Assumptions:
→ Customers are considered new if they have never been seen before.



Step 2. Write in SQL.

```
SELECT shopper_id,
       DATE_TRUNC( 'month', MIN(purchased_at) ) as first_month
FROM transactions
GROUP BY shopper_id
```

Q. How to identify "new customers" in SQL?

A. If their MIN(purchased_at) is within the month.

Plan:
First compute a table of (shopper_id, months) representing shoppers and their first month
Then count the number of rows per month.

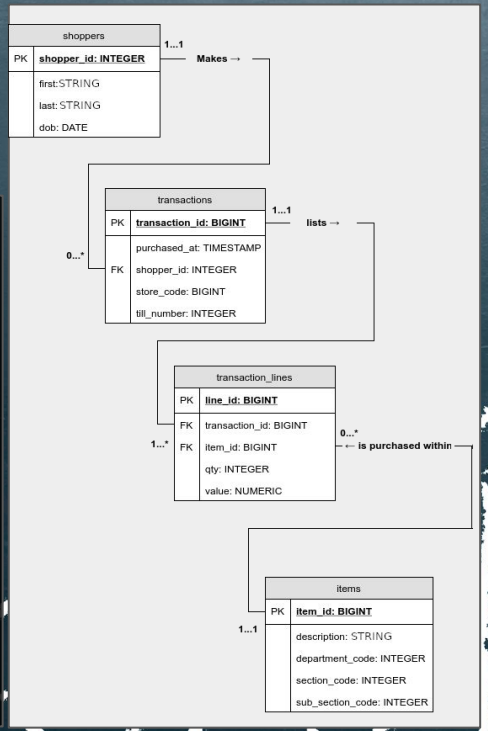
Example 6

Growth
Engagement
Retention rate

Step 1:
Write down a concrete measurable description of the KPI

Engagement, measured as
→ Count of new customers per month
(KPI 6)

KPI 6 Assumptions:
→ Customers are considered new if they have never been seen before.



Step 2. Write in SQL.

```
SELECT date_format(first_month, 'yyyy-MM') as mth, COUNT(*) as ct
FROM (
        SELECT shopper_id,
                DATE_TRUNC( 'month', MIN(purchased_at) ) as first_month
        FROM transactions
        GROUP BY shopper_id
) x
GROUP BY first_month
```

Q. How to identify "new customers" in SQL?

A. If their MIN(purchased_at) is within the month.

Plan:
First compute a table of (shopper_id, months) representing shoppers and their first month
Then count the number of rows per month.

Growth
Engagement
Retention rate

Ok, so returning customers and new customers are two measures of engagement.

A more sophisticated approach is via **cohort retention analysis.**

Example

Active Users:

Jan	Feb	Mar	Apr	May
180	195	200	190	180

Returning customers

50	30	20	0	0

New customers

130	165	180	190	180

Significant problem for most businesses!

Retention / churn is a very common engagement KPI.

Growth
Engagement
Retention rate

Cohort Retention Analysis:
→ Define a cohort

→ Define the period to measure over

Cohort:
→ Group of people with a shared **temporal characteristic**
→ Group of products with a shared **temporal characteristic**

App Launched ↓ % Active users after App Launches →

Cohort	Users	Day 0	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7
Jan 25	1,098	100%	33.9%	23.5%	18.7%	15.9%	16.3%	14.2%	14.5%

People will be assigned to each group based on a characteristic.

→ Typically this is the first interaction date with the company.

→ Or their first interaction with a product following a product launch

Products will be assigned to each group based on a characteristic.

→ Product launch date
→ Advert campaign launch date
→ Email promotion inclusion date

Growth
Engagement
Retention rate

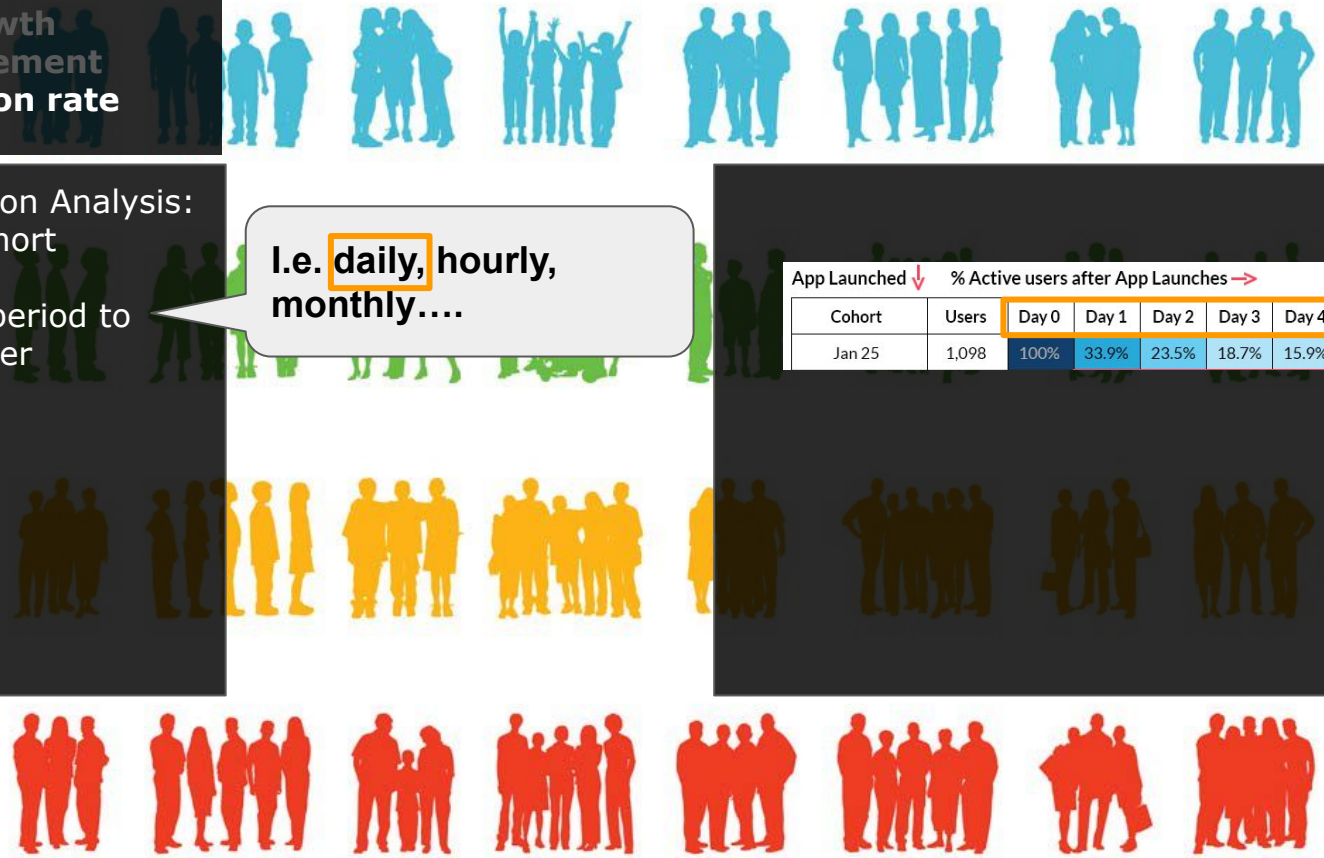
Cohort Retention Analysis:
→ Define a cohort

→ Define the period to measure over

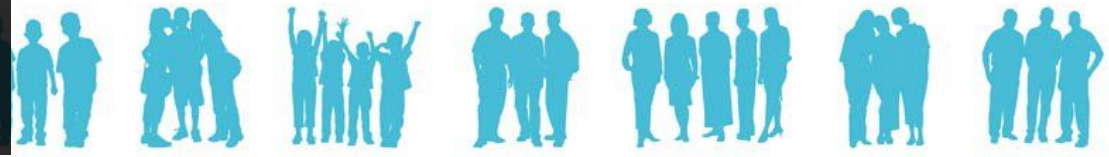
I.e. daily, hourly, monthly….

App Launched ↓ % Active users after App Launches →

Cohort	Users	Day 0	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7
Jan 25	1,098	100%	33.9%	23.5%	18.7%	15.9%	16.3%	14.2%	14.5%



Growth
Engagement
Retention rate



Cohort Retention Analysis:
→ Define a cohort

→ Define the period to measure over

→ Select what to measure per cohort, per day.

can increase on a previous day

App Launched ↓ % Active users after App Launches →

Cohort	Users	Day 0	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7
Jan 25	1,098	100%	33.9%	23.5%	18.7%	15.9%	16.3%	14.2%	14.5%

For retention:
→ **% active users**
→ **% non-lapsed**
→ **...**

For products:
→ % department sales
→ % returns
→ ...

Growth
Engagement
Retention rate

Cohorts performance per time period is then plotted against each other **relative to the initial value of the temporal characteristic.**

Cohort Retention Analysis:
→ Define a cohort

→ Define the period to measure over

→ Select what to measure per cohort, per day.

Cohort:
→ Group of people with a shared **temporal** characteristic:
their first interaction with a product following its launch

I.e. daily

For retention:
→ % active users

App Launched ↓ % Active users after App Launches →

Cohort	Users	Day 0	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7	Day 8	Day 9	Day 10
Jan 25	1,098	100%	33.9%	23.5%	18.7%	15.9%	16.3%	14.2%	14.5%	Retention over user lifetime		12.1%
Jan 26	1,358	100%	31.1%	18.6%	14.3%	16.0%	14.9%	13.2%	12.9%			
Jan 27	1,257	100%	27.2%	19.6%	14.5%	12.9%	13.4%	13.0%	10.8%	11.4%		
Jan 28	1,587	100%	26.6%	17.9%	14.6%	14.8%	14.9%	13.7%	11.9%			
Jan 29	1,758	100%	26.2%	20.4%	16.9%	14.3%	12.7%	12.5%				
Jan 30	1,624	100%	26.4%	18.1%	13.7%	15.4%	11.8%					
Jan 31	1,541	100%	23.9%	19.6%	15.0%	14.8%						
Feb 01	868	100%	24.7%	16.9%	15.8%							
Feb 02	1,143	Retention over product lifetime		18.5%								
Feb 03	1,253											
All Users	13,487	100%	27.0%	19.2%	15.4%	14.9%	14.0%	13.3%	12.5%	13.1%	12.2%	12.1%

Growth
Engagement
Retention rate

Cohorts performance per time period is then plotted against each other **relative to the initial value of the temporal characteristic.**

Great. So what next?

How to do it in SQL of course...

App Launched ↓ % Active users after App Launches →

Cohort	Users	Day 0	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7	Day 8	Day 9	Day 10
Jan 25	1,098	100%	33.9%	23.5%	18.7%	15.9%	16.3%	14.2%	14.5%	Retention over user lifetime		12.1%
Jan 26	1,358	100%	31.1%	18.6%	14.3%	16.0%	14.9%	13.2%	12.9%			
Jan 27	1,257	100%	27.2%	19.6%	14.5%	12.9%	13.4%	13.0%	10.8%	11.4%		
Jan 28	1,587	100%	26.6%	17.9%	14.6%	14.8%	14.9%	13.7%	11.9%			
Jan 29	1,758	100%	26.2%	20.4%	16.9%	14.3%	12.7%	12.5%				
Jan 30	1,624	100%	26.4%	18.1%	13.7%	15.4%	11.8%					
Jan 31	1,541	100%	23.9%	19.6%	15.0%	14.8%						
Feb 01	868	100%	24.7%	16.9%	15.8%							
Feb 02	1,143	Retention over product lifetime		18.5%								
Feb 03	1,253											
All Users	13,487	100%	27.0%	19.2%	15.4%	14.9%	14.0%	13.3%	12.5%	13.1%	12.2%	12.1%

Growth

Engagement Retention rate

This is a more complex query.

COMMON STRUCTURE. A table of data.

Expanding Tables of Data

Could create a table like in the picture.

BUT WHAT HAPPENS TOMORROW? Add a column?

New table per day?

App Launched ↓ % Active users after App Launches →

Cohort	Users	Day 0	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7	Day 8	Day 9	Day 10
Jan 25	1,098	100%	33.9%	23.5%	18.7%	15.9%	16.3%	14.2%	14.5%	Retention over user lifetime		12.1%
Jan 26	1,358	100%	31.1%	18.6%	14.3%	16.0%	14.9%	13.2%	12.9%			
Jan 27	1,257	100%	27.2%	19.6%	14.5%	12.9%	13.4%	13.0%	10.8%	11.4%		
Jan 28	1,587	100%	26.6%	17.9%	14.6%	14.8%	14.9%	13.7%	11.9%			
Jan 29	1,758	100%	26.2%	20.4%	16.9%	14.3%	12.7%	12.5%				
Jan 30	1,624	100%	26.4%	18.1%	13.7%	15.4%	11.8%					
Jan 31	1,541	100%	23.9%	19.6%	15.0%	14.8%						
Feb 01	868	100%	24.7%	16.9%	15.8%							
Feb 02	1,143	Retention over product lifetime	18.5%									
Feb 03	1,253											
All Users	13,487	100%	27.0%	19.2%	15.4%	14.9%	14.0%	13.3%	12.5%	13.1%	12.2%	12.1%

Growth

Engagement Retention rate

Expanding Tables of Data

This is a more complex query.

COMMON STRUCTURE. A table of data.

Data in a table format is represented in a RBMS as tuples of:

(row_val, col_val, cell_val)

App Launched ↓ % Active users after App Launches →

Cohort	Users	Day 0	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7	Day 8	Day 9	Day 10
Jan 25	1,098	100%	33.9%	23.5%	18.7%	15.9%	16.3%	14.2%	14.5%	Retention over user lifetime		12.1%
Jan 26	1,358	100%	31.1%	18.6%	14.3%	16.0%	14.9%	13.2%	12.9%			
Jan 27	1,257	100%	27.2%	19.6%	14.5%	12.9%	13.4%	13.0%	10.8%	11.4%		
Jan 28	1,587	100%	26.6%	17.9%	14.6%	14.8%	14.9%	13.7%	11.9%			
Jan 29	1,758	100%	26.2%	20.4%	16.9%	14.3%	12.7%	12.5%				
Jan 30	1,624	100%	26.4%	18.1%	13.7%	15.4%	11.8%					
Jan 31	1,541	100%	23.9%	19.6%	15.0%	14.8%						
Feb 01	868	100%	24.7%	16.9%	15.8%							
Feb 02	1,143	Retention over product lifetime		18.5%								
Feb 03	1,253											
All Users	13,487	100%	27.0%	19.2%	15.4%	14.9%	14.0%	13.3%	12.5%	13.1%	12.2%	12.1%

(row_val, col_val, cell_val)

('Jan 25', 'Users', 1098)

('Jan 25', 'Day 0', 100.0)

('Jan 25', 'Day 1', 33.9)

.
.
.

('Jan 29', 'Day 5', 12.7)

.
.
.

('All Users', 'Day 10', 12.1)

Growth
Engagement
Retention rate

Expanding Tables of Data

This is a more complex query.

COMMON STRUCTURE. A table of data.

Data in a table format is represented in a RBMS as tuples of:

(row_val, col_val, cell_val)

Exercise: Translate the following tuples into a table:

row_val, col_val, cell_val

'Jack', 'spend', 10.25
'John', 'spend', 6.25
'John', 'visits', 4
'Jack', 'visits', 6

Your turn 4

(row_val, col_val, cell_val)

('Jan 25', 'Users', 1098)
('Jan 25', 'Day 0', 100.0)
('Jan 25', 'Day 1', 33.9)
.
.
.
('Jan 29', 'Day 5', 12.7)
.
.
.
('All Users', 'Day 10', 12.1)

Growth
Engagement
Retention rate

Expanding Tables of Data

This is a more complex query.

COMMON STRUCTURE. A table of data.

Data in a table format is represented in a RBMS as tuples of:

(row_val, col_val, cell_val)

Exercise: Translate the following tuples into a table:

row_val, col_val, cell_val

'Jack', 'spend', 10.25
'John', 'spend', 6.25
'John', 'visits', 4
'Jack', 'visits', 6

	spend	visits
Jack	10.25	6
John	6.25	4

(row_val, col_val, cell_val)

('Jan 25', 'Users', 1098)

('Jan 25', 'Day 0', 100.0)

('Jan 25', 'Day 1', 33.9)

.
.
.

('Jan 29', 'Day 5', 12.7)

.
.
.

('All Users', 'Day 10', 12.1)

Growth
Engagement
Retention rate

Expanding Tables of Data

This is a more complex query.

COMMON STRUCTURE. A table of data.

Data in a table format is represented in a RBMS as tuples of:

(row_val, col_val, cell_val)

Exercise: Translate the following table into tuples:

row_val, col_val, cell_val

	cost (£)	units sold
iphone	800	157
nokia	255	300

Your turn 5

(row_val, col_val, cell_val)

('Jan 25', 'Users', 1098)

('Jan 25', 'Day 0', 100.0)

('Jan 25', 'Day 1', 33.9)

.
.
.

('Jan 29', 'Day 5', 12.7)

.
.
.

('All Users', 'Day 10', 12.1)

Growth
Engagement
Retention rate

Expanding Tables of Data

This is a more complex query.

COMMON STRUCTURE. A table of data.

Data in a table format is represented in a RBMS as tuples of:

(row_val, col_val, cell_val)

Exercise: Translate the following table into tuples:

row_val, col_val, cell_val

'iphone', 'cost (£)', 800
'iphone', 'units sold', 157
'nokia', 'cost (£)', 255
'nokia', 'units sold', 300

	cost (£)	units sold
iphone	800	157
nokia	255	300

(row_val, col_val, cell_val)

('Jan 25', 'Users', 1098)
('Jan 25', 'Day 0', 100.0)
('Jan 25', 'Day 1', 33.9)
.
.
.
('Jan 29', 'Day 5', 12.7)
.
.
.
('All Users', 'Day 10', 12.1)

Growth

Engagement Retention rate

Expanding Tables of Data

This is a more complex query.

COMMON STRUCTURE. A table of data.

Data in a table format is represented in a RBMS as tuples of:

(row_val, col_val, cell_val)

App Launched ↓ % Active users after App Launches →

Cohort	Users	Day 0	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7	Day 8	Day 9	Day 10
Jan 25	1,098	100%	33.9%	23.5%	18.7%	15.9%	16.3%	14.2%	14.5%	Retention over user lifetime		12.1%
Jan 26	1,358	100%	31.1%	18.6%	14.3%	16.0%	14.9%	13.2%	12.9%			
Feb 01	868	100%	24.7%	16.9%	15.8%							
Feb 02	1,143	Retention over product lifetime		18.5%								
Feb 03	1,253											
All Users	13,487	100%	27.0%	19.2%	15.4%	14.9%	14.0%	13.3%	12.5%	13.1%	12.2%	12.1%

This can be easily visualized in Tableau from a tuple representation.

(I'll show you how later today)

(row_val, col_val, cell_val)

('Jan 25', 'Users', 1098)

('Jan 25', 'Day 0', 100.0)

('Jan 25', 'Day 1', 33.9)

.
.
.

('Jan 29', 'Day 5', 12.7)

.
.
.

('All Users', 'Day 10', 12.1)

Growth

Engagement Retention rate

Expanding Tables of Data

OK, so we'll store the table in:

(row_val, col_val, cell_val)

format. **How?**

App Launched ↓ % Active users after App Launches →

Cohort	Users	Day 0	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7	Day 8	Day 9	Day 10
Jan 25	1,098	100%	33.9%	23.5%	18.7%	15.9%	16.3%	14.2%	14.5%	Retention over user lifetime		12.1%
Jan 26	1,358	100%	31.1%	18.6%	14.3%	16.0%	14.9%	13.2%	12.9%			
Jan 27	1,257	100%	27.2%	19.6%	14.5%	12.9%	13.4%	13.0%	10.8%	11.4%		
Jan 28	1,587	100%	26.6%	17.9%	14.6%	14.8%	14.9%	13.7%	11.9%			
Jan 29	1,758	100%	26.2%	20.4%	16.9%	14.3%	12.7%	12.5%				
Jan 30	1,624	100%	26.4%	18.1%	13.7%	15.4%	11.8%					
Jan 31	1,541	100%	23.9%	19.6%	15.0%	14.8%						
Feb 01	868	100%	24.7%	16.9%	15.8%							
Feb 02	1,143	Retention over product lifetime		18.5%								
Feb 03	1,253											
All Users	13,487	100%	27.0%	19.2%	15.4%	14.9%	14.0%	13.3%	12.5%	13.1%	12.2%	12.1%

Multiple steps:

1) Assign customers to cohorts.

2) Count the number of customers active in each subsequent period.

3) Convert the counts to percents

4) Count the number of users per cohort. Add column to table.

5) Compute "All Users" row. Add row to table.

Growth

Engagement Retention rate

Expanding Tables of Data

OK, so we'll store the table in:

(row_val, col_val, cell_val)

format. **How?**

cohort period value
(Jan 25, Day 0, 100%)

App Launched ↓ % Active users after App Launches →

Cohort	Users	Day 0	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7	Day 8	Day 9	Day 10
Jan 25	1,098	100%	33.9%	23.5%	18.7%	15.9%	16.3%	14.2%	14.5%	Retention over user lifetime		12.1%
Jan 26	1,358	100%	31.1%	18.6%	14.3%	16.0%	14.9%	13.2%	12.9%			
Jan 27	1,257	100%	27.2%	19.6%	14.5%	12.9%	13.4%	13.0%	10.8%	11.4%		
Jan 28	1,587	100%	26.6%	17.9%	14.6%	14.8%	14.9%	13.7%	11.9%			
Jan 29	1,758	100%	26.2%	20.4%	16.9%	14.3%	12.7%	12.5%				
Jan 30	1,624	100%	26.4%	18.1%	13.7%	15.4%	11.8%					
Jan 31	1,541	100%	23.9%	19.6%	15.0%	14.8%						
Feb 01	868	100%	24.7%	16.9%	15.8%							
Feb 02	1,143	Retention over product lifetime		18.5%								
Feb 03	1,253											
All Users	13,487	100%	27.0%	19.2%	15.4%	14.9%	14.0%	13.3%	12.5%	13.1%	12.2%	12.1%

Multiple steps:

1) Assign customers to cohorts.

2) Count the number of customers active in each subsequent period.

3) Convert the counts to percents

4) Count the number of users per cohort. Add column to table.

5) Compute "All Users" row. Add row to table.

Growth

Engagement Retention rate

Expanding Tables of Data

OK, so we'll store the table in:

(row_val, col_val, cell_val)

format. **How?**

App Launched ↓ % Active users after App Launches →

Cohort	Users	Day 0	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7	Day 8	Day 9	Day 10
Jan 25	1,098	100%	33.9%	23.5%	18.7%	15.9%	16.3%	14.2%	14.5%	Retention over user lifetime		12.1%
Jan 26	1,358	100%	31.1%	18.6%	14.3%	16.0%	14.9%	13.2%	12.9%			
Jan 27	1,257	100%	27.2%	19.6%	14.5%	12.9%	13.4%	13.0%	10.8%	11.4%		
Jan 28	1,587	100%	26.6%	17.9%	14.6%	14.8%	14.9%	13.7%	11.9%			
Jan 29	1,758	100%	26.2%	20.4%	16.9%	14.3%	12.7%	12.5%				
Jan 30	1,624	100%	26.4%	18.1%	13.7%	15.4%	11.8%					
Jan 31	1,541	100%	23.9%	19.6%	15.0%	14.8%						
Feb 01	868	100%	24.7%	16.9%	15.8%							
Feb 02	1,143	Retention over product lifetime		18.5%								
Feb 03	1,253											
All Users	13,487	100%	27.0%	19.2%	15.4%	14.9%	14.0%	13.3%	12.5%	13.1%	12.2%	12.1%

Multiple steps:

1) Assign customers to cohorts.

2) Count the number of customers active in each subsequent period.

3) Convert the counts to percents

4) Count the number of users per cohort (actually will have already been done as part of 3). Add column to table.

5) Compute "All Users" row. Add row to table.

Growth

Engagement Retention rate

Expanding Tables of Data

OK, so we'll store the table in:

(row_val, col_val, cell_val)

format. **How?**

App Launched ↓ % Active users after App Launches →

Cohort	Users	Day 0	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7	Day 8	Day 9	Day 10
Jan 25	1,098	100%	33.9%	23.5%	18.7%	15.9%	16.3%	14.2%	14.5%	Retention over user lifetime		12.1%
Jan 26	1,358	100%	31.1%	18.6%	14.3%	16.0%	14.9%	13.2%	12.9%			
Jan 27	1,257	100%	27.2%	19.6%	14.5%	12.9%	13.4%	13.0%	10.8%	11.4%		
Jan 28	1,587	100%	26.6%	17.9%	14.6%	14.8%	14.9%	13.7%	11.9%			
Jan 29	1,758	100%	26.2%	20.4%	16.9%	14.3%	12.7%	12.5%				
Jan 30	1,624	100%	26.4%	18.1%	13.7%	15.4%	11.8%					
Jan 31	1,541	100%	23.9%	19.6%	15.0%	14.8%						
Feb 01	868	100%	24.7%	16.9%	15.8%							
Feb 02	1,143	Retention over product lifetime		18.5%								
Feb 03	1,253											
All Users	13,487	100%	27.0%	19.2%	15.4%	14.9%	14.0%	13.3%	12.5%	13.1%	12.2%	12.1%

Multiple steps:

1) Assign customers to cohorts.

2) Count the number of customers active in each subsequent period.

3) Convert the counts to percents

4) Count the number of users per cohort. Add column to table.

5) Compute "All Users" row. Add row to table.

Growth

Engagement Retention rate

Expanding Tables of Data

We're going to stop here to keep things simple in this lecture (cells are then all percentages)

Feel free to try Step 5 yourself.

App Launched ↓ % Active users after App Launches →

Cohort	Users	Day 0	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7	Day 8	Day 9	Day 10
Jan 25	1,098	100%	33.9%	23.5%	18.7%	15.9%	16.3%	14.2%	14.5%	Retention over user lifetime		12.1%
Jan 26	1,358	100%	31.1%	18.6%	14.3%	16.0%	14.9%	13.2%	12.9%			
Jan 27	1,257	100%	27.2%	19.6%	14.5%	12.9%	13.4%	13.0%	10.8%	11.4%		
Jan 28	1,587	100%	26.6%	17.9%	14.6%	14.8%	14.9%	13.7%	11.9%			
Jan 29	1,758	100%	26.2%	20.4%	16.9%	14.3%	12.7%	12.5%				
Jan 30	1,624	100%	26.4%	18.1%	13.7%	15.4%	11.8%					
Jan 31	1,541	100%	23.9%	19.6%	15.0%	14.8%						
Feb 01	868	100%	24.7%	16.9%	15.8%							
Feb 02	1,143	Retention over product lifetime		18.5%								
Feb 03	1,253											
All Users	13,487	100%	27.0%	19.2%	15.4%	14.9%	14.0%	13.3%	12.5%	13.1%	12.2%	12.1%

Multiple steps:

1) Assign customers to cohorts.

2) Count the number of customers active in each subsequent period.

3) Convert the counts to percents

4) Count the number of users per cohort. Add column to table.

5) Compute "All Users" row. Add row to table.

Example 7

Cohort Retention Analysis:
Supermarket customer retention

shoppers	
PK	**shopper_id: INTEGER**
	first: STRING
	last: STRING
	dob: DATE

transactions	
PK	**transaction_id: BIGINT**
	purchased_at: TIMESTAMP
FK	shopper_id: INTEGER
	store_code: BIGINT
	till_number: INTEGER

transaction_lines	
PK	**line_id: BIGINT**
FK	transaction_id: BIGINT
FK	item_id: BIGINT
	qty: INTEGER
	value: NUMERIC

items	
PK	**item_id: BIGINT**
	description: STRING
	department_code: INTEGER
	section_code: INTEGER
	sub_section_code: INTEGER

First, fix our definitions

Cohort Retention Analysis:

→ Define a cohort

→ Define the period to measure over

→ Select what to measure per cohort, per period.

Cohort:
→ Group of people with a shared **temporal** characteristic:
their first purchase date

Monthly

For retention:
→ % active users

Example 7

shoppers	
PK	**shopper_id: INTEGER**
	first: STRING
	last: STRING
	dob: DATE

transactions	
PK	**transaction_id: BIGINT**
	purchased_at: TIMESTAMP
FK	shopper_id: INTEGER
	store_code: BIGINT
	till_number: INTEGER

transaction_lines	
PK	**line_id: BIGINT**
FK	transaction_id: BIGINT
FK	item_id: BIGINT
	qty: INTEGER
	value: NUMERIC

items	
PK	**item_id: BIGINT**
	description: STRING
	department_code: INTEGER
	section_code: INTEGER
	sub_section_code: INTEGER

Cohort Retention Analysis:
Supermarket customer retention

Cohort characteristic:
1st purchase date

Period to measure over:
Monthly

Measure, per cohort, per period:
% active users

Multiple steps:

1) Assign customers to cohorts.

2) Count the number of customers active in each subsequent period.

3) Convert the counts to percents

4) Count the number of users per cohort. Add column to table.

```
-- let's make a table of all shopper_ids and their cohort, denoted by their first month
```

```
Output table headings:
shopper_id (INT),
cohort_date(DATE)
```

Your turn 6

We are first going to do this in stages, creating temporary tables.

Remember the syntax?

CREATE TABLE cohort_assignment AS ….. <your select statement?

Example 7

shoppers	
PK	**shopper_id: INTEGER**
	first: STRING
	last: STRING
	dob: DATE

transactions	
PK	**transaction_id: BIGINT**
	purchased_at: TIMESTAMP
FK	shopper_id: INTEGER
	store_code: BIGINT
	till_number: INTEGER

transaction_lines	
PK	**line_id: BIGINT**
FK	transaction_id: BIGINT
FK	item_id: BIGINT
	qty: INTEGER
	value: NUMERIC

items	
PK	**item_id: BIGINT**
	description: STRING
	department_code: INTEGER
	section_code: INTEGER
	sub_section_code: INTEGER

Cohort Retention Analysis:
Supermarket customer retention

Cohort characteristic:
1st purchase date

Period to measure over:
Monthly

Measure, per cohort, per period:
% active users

Multiple steps:

1) Assign customers to cohorts.

2) Count the number of customers active in each subsequent period.

3) Convert the counts to percents

4) Count the number of users per cohort. Add column to table.

```
-- let's make a table of all shopper_ids and their cohort, denoted by their first month

CREATE TABLE cohort_assignment AS
SELECT shopper_id,
       DATE_TRUNC('month', MIN(purchased_at))::DATE as cohort_date
FROM transactions
GROUP BY shopper_id;
```

Example 7

shoppers	
PK	**shopper_id: INTEGER**
	first: STRING
	last: STRING
	dob: DATE

transactions	
PK	**transaction_id: BIGINT**
	purchased_at: TIMESTAMP
FK	shopper_id: INTEGER
	store_code: BIGINT
	till_number: INTEGER

transaction_lines	
PK	**line_id: BIGINT**
FK	transaction_id: BIGINT
FK	item_id: BIGINT
	qty: INTEGER
	value: NUMERIC

items	
PK	**item_id: BIGINT**
	description: STRING
	department_code: INTEGER
	section_code: INTEGER
	sub_section_code: INTEGER

Cohort Retention Analysis:
Supermarket customer retention

Cohort characteristic:
1st purchase date

Period to measure over:
Monthly

Measure, per cohort, per period:
% active users

Multiple steps:

1) Assign customers to cohorts.

2) Count the number of customers active in each subsequent period.

3) Convert the counts to percents

4) Count the number of users per cohort. Add column to table.

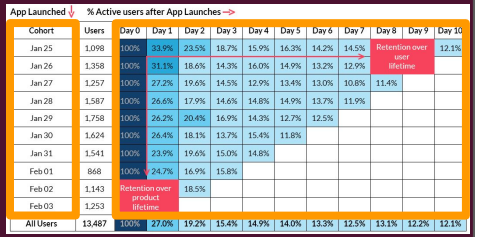
```
-- Logically this can be done by:
1) Extending all transaction records with the associated shopper's cohort date.
2) Compute a new column that transforms the records purchased_at date → relative period (month)
   since the shopper's cohort date.
3) Putting records in buckets based on the cohort_date and relative_period and working how many
   shoppers were active
```

Your turn 7

```
Output table headings:

cohort_date(DATE),
relative_period (INT),
active_ct (INT)
```

cohort period active_value
(2017-01-01, 1, 10)



Cohort	Users	Day 0	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7	Day 8	Day 9	Day 10
Jan 25	1,098	100%	33.9%	23.5%	18.7%	15.9%	16.3%	14.2%	14.5%			12.1%
Jan 26	1,358	100%	31.1%	18.6%	14.3%	16.0%	14.9%	13.2%	12.9%			
Jan 27	1,257	100%	27.2%	19.6%	14.5%	12.9%	13.4%	13.0%	10.8%	11.4%		
Jan 28	1,587	100%	26.6%	17.9%	14.6%	14.8%	14.9%	13.7%	11.9%			
Jan 29	1,758	100%	26.2%	20.4%	16.9%	14.3%	12.7%	12.5%				
Jan 30	1,624	100%	26.4%	18.1%	13.7%	15.4%	11.8%					
Jan 31	1,541	100%	23.9%	19.6%	15.0%	14.8%						
Feb 01	868	100%	24.7%	16.9%	15.8%							
Feb 02	1,143	100%	18.5%									
Feb 03	1,253											
All Users	13,487	100%	27.0%	19.2%	15.4%	14.9%	14.0%	13.3%	12.5%	13.1%	12.2%	12.1%

Example 7

shoppers	
PK	**shopper_id: INTEGER**
	first: STRING
	last: STRING
	dob: DATE

transactions	
PK	**transaction_id: BIGINT**
	purchased_at: TIMESTAMP
FK	shopper_id: INTEGER
	store_code: BIGINT
	till_number: INTEGER

transaction_lines	
PK	**line_id: BIGINT**
FK	transaction_id: BIGINT
FK	item_id: BIGINT
	qty: INTEGER
	value: NUMERIC

items	
PK	**item_id: BIGINT**
	description: STRING
	department_code: INTEGER
	section_code: INTEGER
	sub_section_code: INTEGER

Cohort Retention Analysis:
Supermarket customer retention

Cohort characteristic:
1st purchase date

Period to measure over:
Monthly

Measure, per cohort, per period:
% active users

Multiple steps:

1) Assign customers to cohorts.

2) Count the number of customers active in each subsequent period.

3) Convert the counts to percents

4) Count the number of users per cohort. Add column to table.

```
-- Logically this can be done by:
1) Extending all transaction records with the associated shopper's cohort date.


CREATE TABLE cohort_mth_cts AS
SELECT
FROM transactions
JOIN cohort_assignment
USING (shopper_id);
```

Example 7

shoppers	
PK	shopper_id: INTEGER
	first: STRING
	last: STRING
	dob: DATE

transactions	
PK	transaction_id: BIGINT
	purchased_at: TIMESTAMP
FK	shopper_id: INTEGER
	store_code: BIGINT
	till_number: INTEGER

transaction_lines	
PK	line_id: BIGINT
FK	transaction_id: BIGINT
FK	item_id: BIGINT
	qty: INTEGER
	value: NUMERIC

items	
PK	item_id: BIGINT
	description: STRING
	department_code: INTEGER
	section_code: INTEGER
	sub_section_code: INTEGER

Cohort Retention Analysis:
Supermarket customer retention

Cohort characteristic:
1st purchase date

Period to measure over:
Monthly

Measure, per cohort, per period:
% active users

Multiple steps:

1) Assign customers to cohorts.

2) Count the number of customers active in each subsequent period.

3) Convert the counts to percents

4) Count the number of users per cohort. Add column to table.

```
-- Logically this can be done by:
1) Extending all transaction records with the associated shopper's cohort date.
2) Compute a new column that transforms the records purchased_at date → relative period
   (month) since the shopper's cohort date.

CREATE TABLE cohort_mth_cts AS
SELECT cohort_date,
       MONTHS_BETWEEN(
              DATE_TRUNC('month', purchased_at),
              DATE_TRUNC('month', cohort_date)
       ) as relative_period
FROM transactions
JOIN cohort_assignment
USING (shopper_id)
```

Example 7

shoppers	
PK	**shopper_id: INTEGER**
	first: STRING
	last: STRING
	dob: DATE

transactions	
PK	**transaction_id: BIGINT**
	purchased_at: TIMESTAMP
FK	shopper_id: INTEGER
	store_code: BIGINT
	till_number: INTEGER

transaction_lines	
PK	**line_id: BIGINT**
FK	transaction_id: BIGINT
FK	item_id: BIGINT
	qty: INTEGER
	value: NUMERIC

items	
PK	**item_id: BIGINT**
	description: STRING
	department_code: INTEGER
	section_code: INTEGER
	sub_section_code: INTEGER

Cohort Retention Analysis:
Supermarket customer retention

Cohort characteristic:
1st purchase date

Period to measure over:
Monthly

Measure, per cohort, per period:
% active users

Multiple steps:

1) Assign customers to cohorts.

2) Count the number of customers active in each subsequent period.

3) Convert the counts to percents

4) Count the number of users per cohort. Add column to table.

```
-- Logically this can be done by:
1) Extending all transaction records with the associated shopper's cohort date.
2) Compute a new column that transforms the records purchased_at date → relative period
   (month) since the shopper's cohort date.
3) putting records in buckets based on the cohort_date and relative_period and working how many
   shoppers were active

CREATE TABLE cohort_mth_cts AS
SELECT cohort_date,
       MONTHS_BETWEEN(
              DATE_TRUNC('month', purchased_at),
              DATE_TRUNC('month', cohort_date)
       ) as relative_period,
       COUNT(DISTINCT shopper_id) as active_ct
FROM transactions
JOIN cohort_assignment
USING (shopper_id)
GROUP BY cohort_date, relative_period;
```

Example 7

shoppers	
PK	**shopper_id: INTEGER**
	first: STRING
	last: STRING
	dob: DATE

transactions	
PK	**transaction_id: BIGINT**
	purchased_at: TIMESTAMP
FK	shopper_id: INTEGER
	store_code: BIGINT
	till_number: INTEGER

transaction_lines	
PK	**line_id: BIGINT**
FK	transaction_id: BIGINT
FK	item_id: BIGINT
	qty: INTEGER
	value: NUMERIC

items	
PK	**item_id: BIGINT**
	description: STRING
	department_code: INTEGER
	section_code: INTEGER
	sub_section_code: INTEGER

Cohort Retention Analysis:
Supermarket customer retention

Cohort characteristic:
1st purchase date

Period to measure over:
Monthly

Measure, per cohort, per period:
% active users

Multiple steps:

1) Assign customers to cohorts.

2) Count the number of customers active in each subsequent period.

3) Convert the counts to percents

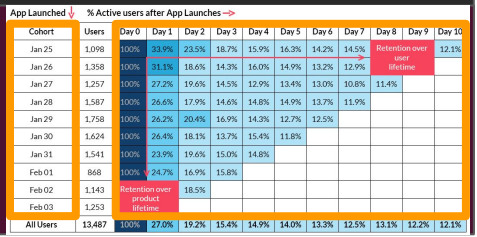
4) Count the number of users per cohort. Add column to table.

Demo. Let's see what we've created in Tableau!

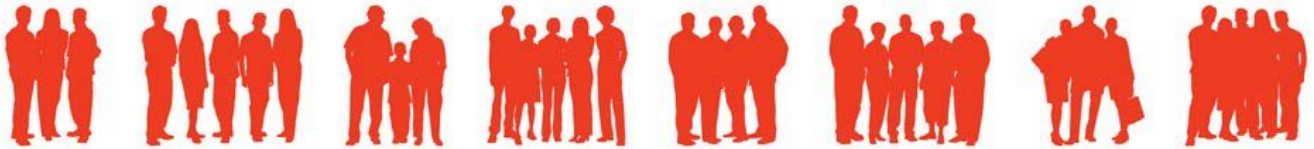
```
CREATE TABLE cohort_mth_cts AS
SELECT cohort_date,
       MONTHS_BETWEEN(
              DATE_TRUNC('month', purchased
              DATE_TRUNC('month', cohort_da
       ) as relative_period,
       COUNT(DISTINCT shopper_id) as active
FROM transactions
JOIN cohort_assignment
USING (shopper_id)
GROUP BY cohort_date, relative_period;
```

cohort, period, active_val
(Jan 25, 1, 0.339)

NOTE: right now we have active counts, rather than %.



Cohort	Users	Day 0	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7	Day 8	Day 9	Day 10
Jan 25	1,098	100%	33.9%	23.5%	18.7%	15.9%	16.3%	14.2%	14.5%			12.1%
Jan 26	1,358	100%	31.1%	18.6%	14.3%	16.0%	14.9%	13.2%	12.9%			
Jan 27	1,257	100%	27.2%	19.6%	14.5%	12.9%	13.4%	13.0%	10.8%	11.4%		
Jan 28	1,587	100%	26.6%	17.9%	14.6%	14.8%	14.9%	13.7%	11.9%			
Jan 29	1,758	100%	26.2%	20.4%	16.9%	14.3%	12.7%	12.5%				
Jan 30	1,624	100%	26.4%	18.1%	13.7%	15.4%	11.8%					
Jan 31	1,541	100%	23.9%	19.6%	15.0%	14.8%						
Feb 01	868	100%	24.7%	16.9%	15.8%							
Feb 02	1,143	100%	18.5%									
Feb 03	1,253											
All Users	13,487	100%	27.0%	19.2%	15.4%	14.9%	14.0%	13.3%	12.5%	13.1%	12.2%	12.1%



Example 7

shoppers	
PK	shopper_id: INTEGER
	first: STRING
	last: STRING
	dob: DATE

transactions	
PK	transaction_id: BIGINT
	purchased_at: TIMESTAMP
FK	shopper_id: INTEGER
	store_code: BIGINT
	till_number: INTEGER

transaction_lines	
PK	line_id: BIGINT
FK	transaction_id: BIGINT
FK	item_id: BIGINT
	qty: INTEGER
	value: NUMERIC

items	
PK	item_id: BIGINT
	description: STRING
	department_code: INTEGER
	section_code: INTEGER
	sub_section_code: INTEGER

Cohort Retention Analysis:
Supermarket customer retention

Cohort characteristic:
1st purchase date

Period to measure over:
Monthly

Measure, per cohort, per period:
% active users

Multiple steps:

1) Assign customers to cohorts.

2) Count the number of customers active in each subsequent period.

3) Convert the counts to percents

4) Count the number of users per cohort. Add column to table.

```
-- We could not normalize directly as we need to group by two different things
   numerator:   cohort_date and relative_period
   denominator: cohort_date
```

CODE FROM PREVIOUS STEP

```
CREATE TABLE cohort_mth_cts AS
SELECT cohort_date,
       MONTHS_BETWEEN(
              DATE_TRUNC('month', purchased_at),
              DATE_TRUNC('month', cohort_date)
       ) as relative_period,
       COUNT(DISTINCT shopper_id) as active_ct
FROM transactions
JOIN cohort_assignment
USING (shopper_id)
GROUP BY cohort_date, relative_period;
```

Example 7

shoppers	
PK	**shopper_id: INTEGER**
	first: STRING
	last: STRING
	dob: DATE

transactions	
PK	**transaction_id: BIGINT**
	purchased_at: TIMESTAMP
FK	shopper_id: INTEGER
	store_code: BIGINT
	till_number: INTEGER

transaction_lines	
PK	**line_id: BIGINT**
FK	transaction_id: BIGINT
FK	item_id: BIGINT
	qty: INTEGER
	value: NUMERIC

items	
PK	**item_id: BIGINT**
	description: STRING
	department_code: INTEGER
	section_code: INTEGER
	sub_section_code: INTEGER

Cohort Retention Analysis:
Supermarket customer retention

Cohort characteristic:
1st purchase date

Period to measure over:
Monthly

Measure, per cohort, per period:
% active users

Multiple steps:

1) Assign customers to cohorts.

2) Count the number of customers active in each subsequent period.

3) Convert the counts to percents

4) Count the number of users per cohort. Add column to table.

```
-- We could not normalize directly as we need to group by two different things
   numerator:   cohort_date and relative_period
   denominator: cohort_date
-- SOLUTION: Create a new table with just the denominator and JOIN it to the table
             cohort_mth_cts table, only keeping row for which the denominator logically
             matches with the numerator.

CREATE TABLE cohort_totals AS
SELECT cohort_date, COUNT(DISTINCT shopper_id) AS cohort_total
FROM cohort_assignment
GROUP BY cohort_date;
```

Example 7

shoppers	
PK	shopper_id: INTEGER
	first: STRING
	last: STRING
	dob: DATE

transactions	
PK	transaction_id: BIGINT
	purchased_at: TIMESTAMP
FK	shopper_id: INTEGER
	store_code: BIGINT
	till_number: INTEGER

transaction_lines	
PK	line_id: BIGINT
FK	transaction_id: BIGINT
FK	item_id: BIGINT
	qty: INTEGER
	value: NUMERIC

items	
PK	item_id: BIGINT
	description: STRING
	department_code: INTEGER
	section_code: INTEGER
	sub_section_code: INTEGER

Cohort Retention Analysis:
Supermarket customer retention

Cohort characteristic:
1st purchase date

Period to measure over:
Monthly

Measure, per cohort, per period:
% active users

Multiple steps:

1) Assign customers to cohorts.

2) Count the number of customers active in each subsequent period.

3) Convert the counts to percents

4) Count the number of users per cohort. Add column to table.

```
-- We could not normalize directly as we need to group by two different things
   numerator:   cohort_date and relative_period
   denominator: cohort_date
-- SOLUTION: Create a new table with just the denominator and JOIN it to the table
             cohort_mth_cts table, only keeping row for which the denominator logically
             matches with the numerator.

CREATE TABLE cohort_totals AS
SELECT cohort_date, COUNT(DISTINCT shopper_id) AS cohort_total
FROM cohort_assignment
GROUP BY cohort_date;

CREATE TABLE cohort_mth_percent AS
SELECT cohort_date, relative_period, active_ct / cohort_total as active_percent
FROM cohort_mth_cts
JOIN cohort_totals
USING (cohort_date);
```

Example 7

shoppers	
PK	**shopper_id: INTEGER**
	first: STRING
	last: STRING
	dob: DATE

transactions	
PK	**transaction_id: BIGINT**
	purchased_at: TIMESTAMP
FK	shopper_id: INTEGER
	store_code: BIGINT
	till_number: INTEGER

transaction_lines	
PK	**line_id: BIGINT**
FK	transaction_id: BIGINT
FK	item_id: BIGINT
	qty: INTEGER
	value: NUMERIC

items	
PK	**item_id: BIGINT**
	description: STRING
	department_code: INTEGER
	section_code: INTEGER
	sub_section_code: INTEGER

Cohort Retention Analysis:
Supermarket customer retention

Cohort characteristic:
1st purchase date

Period to measure over:
Monthly

Measure, per cohort, per period:
% active users

Multiple steps:

1) Assign customers to cohorts.

2) Count the number of customers active in each subsequent period.

3) Convert the counts to percents

4) Count the number of users per cohort. Add column to table.

```
-- We could not normalize directly as we ne
   numerator:   cohort_date and relative_pe
   denominator: cohort_date
-- SOLUTION: Create a new table with just t
             cohort_mth_cts table, only kee
             matches with the numerator.
```

`q.cohort_mth_percent`

cohort, period, active_val
(Jan 25, 1, 0.339)

App Launched ↓ | % Active users after App Launches →

Cohort	Users	Day 0	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7	Day 8	Day 9	Day 10
Jan 25	1,098	100%	33.9%	23.5%	18.7%	15.9%	16.3%	14.2%	14.5%	Retention over user lifetime		12.1%
Jan 26	1,358	100%	31.1%	18.6%	14.3%	16.0%	14.9%	13.2%	12.9%			
Jan 27	1,257	100%	27.2%	19.6%	14.5%	12.9%	13.4%	13.0%	10.8%	11.4%		
Jan 28	1,587	100%	26.6%	17.9%	14.6%	14.8%	14.9%	13.7%	11.9%			
Jan 29	1,758	100%	26.2%	20.4%	16.9%	14.3%	12.7%	12.5%				
Jan 30	1,624	100%	26.4%	18.1%	13.7%	15.4%	11.8%					
Jan 31	1,541	100%	23.9%	19.6%	15.0%	14.8%						
Feb 01	868	100%	24.7%	16.9%	15.8%							
Feb 02	1,143	Retention over product lifetime	18.5%									
Feb 03	1,253											
All Users	13,487	100%	27.0%	19.2%	15.4%	14.9%	14.0%	13.3%	12.5%	13.1%	12.2%	12.1%

```
CREATE TABLE cohort_totals AS
SELECT cohort_date, COUNT(DISTINCT shopper_id) AS cohort_total
FROM cohort_assignment
GROUP BY cohort_date;
```

Your turn 8

```
CREATE TABLE cohort_mth_percent AS
SELECT cohort_date, relative_period, active_ct / cohort_total as active_percent
FROM cohort_mth_cts
JOIN cohort_totals
USING (cohort_date);
```

Expanding table structure:

(row_val, col_val, cell_val)

Multiple steps:

1) Assign customers to cohorts.

2) Count the number of customers active in each subsequent period.

3) Convert the counts to percents

4) Count the number of users per cohort. Add column to table.

Adding column

To add a new **column** we need to add extra tuples.

One tuple per row value.

All tuples have same col_val, the name of the new column.

(row_id, new_col_id, cell_val)

Fixed values

App Launched ↓ % Active users after App Launches →

Cohort	Users	Day 0	Day 1	Day 2	Day 3	Day 4	Day 5	Day 6	Day 7	Day 8	Day 9	Day 10
Jan 25	1,542	100%	33.9%	23.5%	18.7%	15.9%	16.3%	14.2%	14.5%	Retention over user lifetime		12.1%
Jan 26	1,358	100%	31.1%	18.6%	14.3%	16.0%	14.9%	13.2%	12.9%			
Jan 27	1,257	100%	27.2%	19.6%	14.5%	12.9%	13.4%	13.0%	10.8%	11.4%		
Jan 28	1,587	100%	26.6%	17.9%	14.6%	14.8%	14.9%	13.7%	11.9%			
Jan 29	1,758	100%	26.2%	20.4%	16.9%	14.3%	12.7%	12.5%				
Jan 30	1,624	100%	26.4%	18.1%	13.7%	15.4%	11.8%					
Jan 31	1,541	100%	23.9%	19.6%	15.0%	14.8%						
Feb 01	868	100%	24.7%	16.9%	15.8%							
Feb 02	1,143	Retention over product lifetime	18.5%									
Feb 03	1,253											
All Users	13,463	100%	27.0%	19.2%	15.4%	14.9%	14.0%	13.3%	12.5%	13.1%	12.2%	12.1%

row_id = cohort_date
col_id = 'total'
cell_val = total shoppers for the cohort_date

We've seen these values before….

Example 7

Expanding table structure:

(row_val, col_val, cell_val)

Column value will be fixed to name of new column being added.

Multiple steps:

1) Assign customers to cohorts.

2) Count the number of customers active in each subsequent period.

3) Convert the counts to percents

4) Count the number of users per cohort. Add column to table.

Our Example Cont.

RECALL:

```
CREATE TABLE cohort_totals AS
SELECT cohort_date, COUNT(DISTINCT shopper_id) AS cohort_total
FROM cohort_assignment
GROUP BY cohort_date;
```

So the extra rows we want are the same as in cohort_totals with an extra fixed column...

```
SELECT
FROM cohort_totals
```

Example 7

Expanding table structure:

(row_val, col_val, cell_val)

Column value will be fixed to name of new column being added.

Multiple steps:

1) Assign customers to cohorts.

2) Count the number of customers active in each subsequent period.

3) Convert the counts to percents

4) Count the number of users per cohort. Add column to table.

Our Example Cont.

RECALL:

```
CREATE TABLE cohort_totals AS
SELECT cohort_date, COUNT(DISTINCT shopper_id) AS cohort_total
FROM cohort_assignment
GROUP BY cohort_date;
```

So the extra rows we want are the same as in cohort_totals with an extra fixed column...

```
SELECT cohort_date, 'total'::STRING, cohort_total
FROM cohort_totals
```

Example 7

Expanding table structure:

(row_val, col_val, cell_val)

Column value will be fixed to name of new column being added.

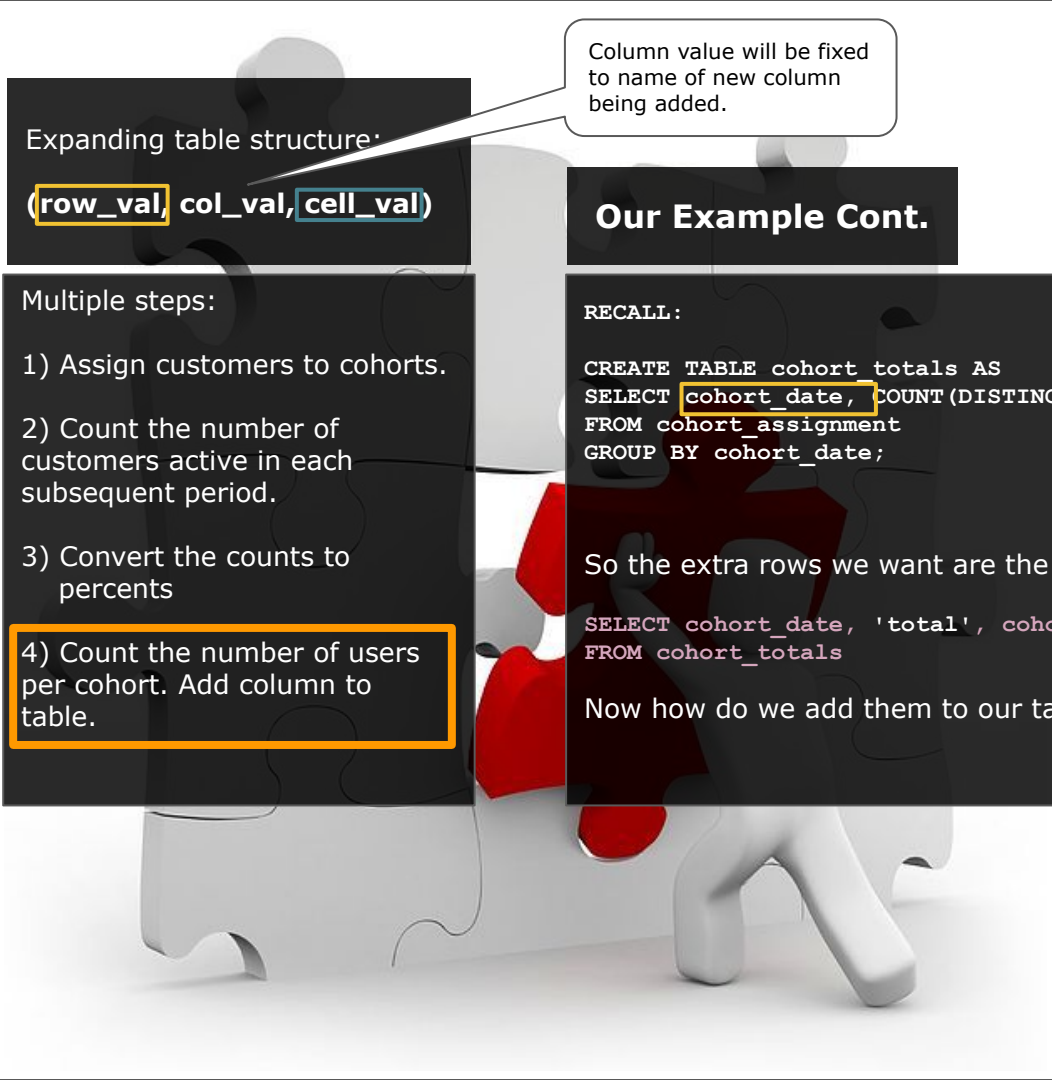
Multiple steps:

1) Assign customers to cohorts.

2) Count the number of customers active in each subsequent period.

3) Convert the counts to percents

4) Count the number of users per cohort. Add column to table.



Our Example Cont.

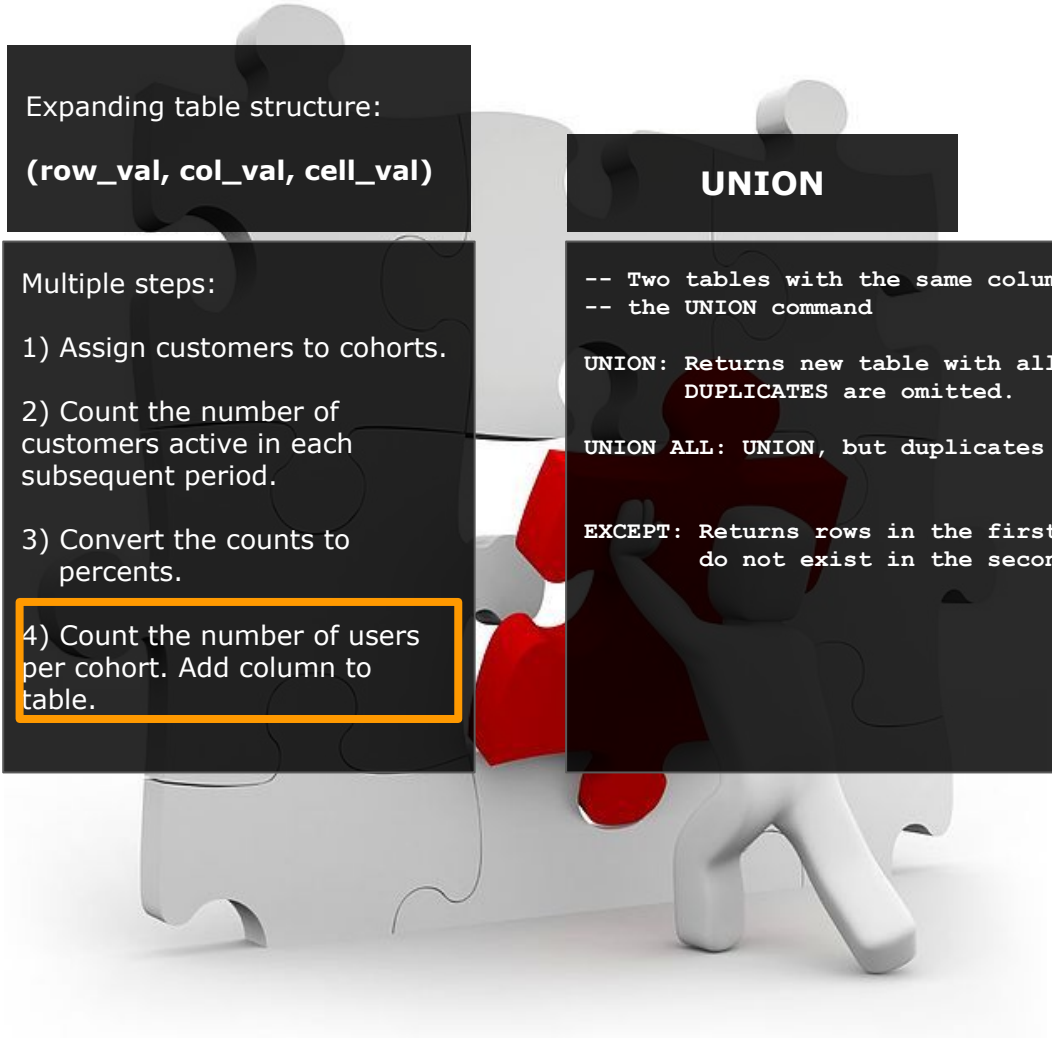
RECALL:

```
CREATE TABLE cohort_totals AS
SELECT cohort_date, COUNT(DISTINCT shopper_id) AS cohort_total
FROM cohort_assignment
GROUP BY cohort_date;
```

So the extra rows we want are the same as in `cohort_totals` with an extra fixed column...

```
SELECT cohort_date, 'total', cohort_total
FROM cohort_totals
```

Now how do we add them to our table `cohort_mth_percent` (represented in tuple format)?



Expanding table structure:

(row_val, col_val, cell_val)

Multiple steps:

1) Assign customers to cohorts.

2) Count the number of customers active in each subsequent period.

3) Convert the counts to percents.

4) Count the number of users per cohort. Add column to table.

UNION

```
-- Two tables with the same columns can be merged using
-- the UNION command

UNION: Returns new table with all rows from both tables.
       DUPLICATES are omitted.

UNION ALL: UNION, but duplicates kept.


EXCEPT: Returns rows in the first table that
        do not exist in the second table.
```

Example 7

Expanding table structure:

(row_val, col_val, cell_val)

Column value will be fixed to name of new column being added.

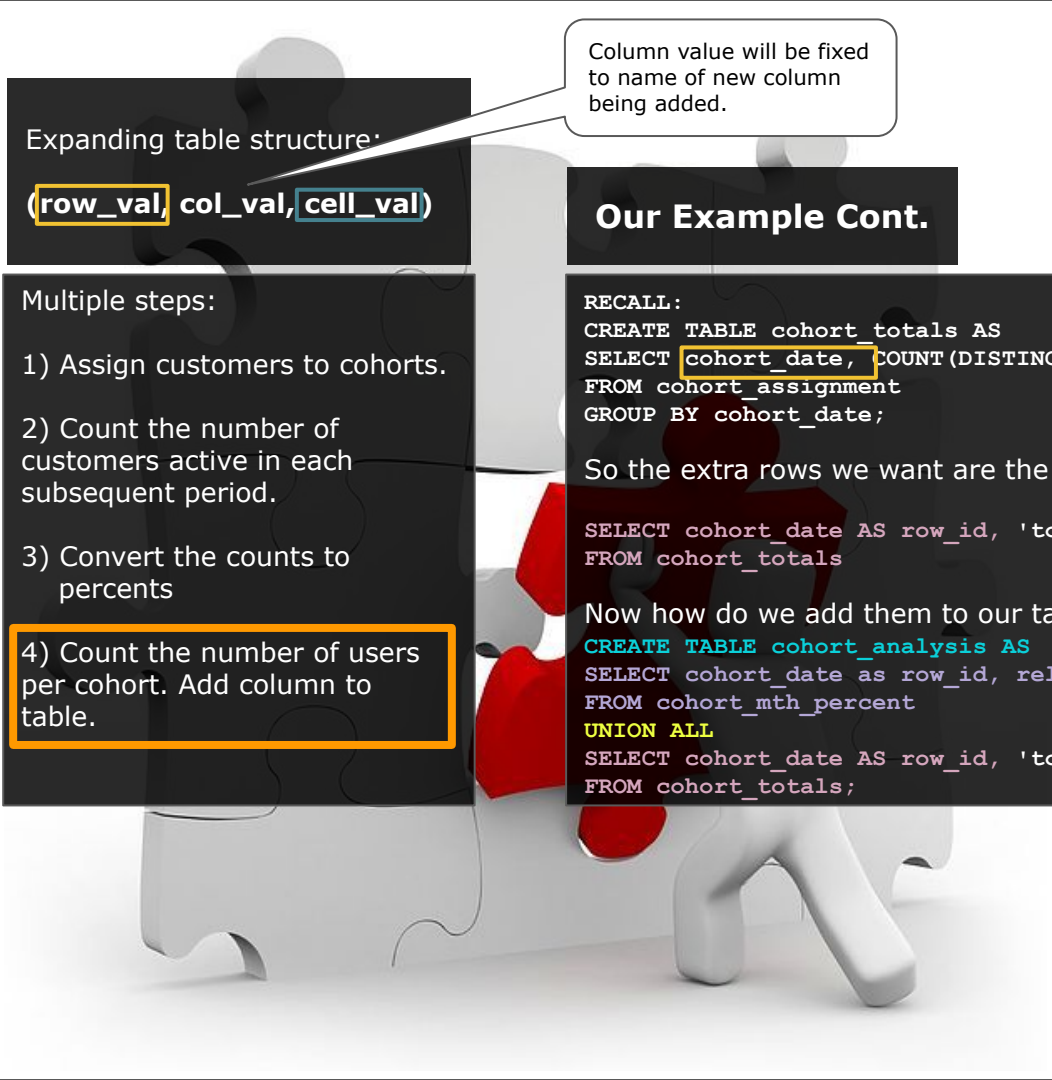
Multiple steps:

1) Assign customers to cohorts.

2) Count the number of customers active in each subsequent period.

3) Convert the counts to percents

4) Count the number of users per cohort. Add column to table.



Our Example Cont.

```
RECALL:
CREATE TABLE cohort_totals AS
SELECT cohort_date, COUNT(DISTINCT shopper_id) AS cohort_total
FROM cohort_assignment
GROUP BY cohort_date;
```

So the extra rows we want are the same as in cohort_totals with an extra fixed column...

```
SELECT cohort_date AS row_id, 'total' AS col_id, cohort_total AS val
FROM cohort_totals
```

Now how do we add them to our table **cohort_mth_percent** (represented in tuple format)?

```
CREATE TABLE cohort_analysis AS
SELECT cohort_date as row_id, relative_period::STRING as col_id, active_percent as val
FROM cohort_mth_percent
UNION ALL
SELECT cohort_date AS row_id, 'total'::STRING AS col_id, cohort_total AS val
FROM cohort_totals;
```

Example 7

Expanding table structure:

(**row_val**, **col_val**, **cell_val**)

Column value will be fixed to name of new column being added.

Multiple steps:

1) Assign customers to cohorts.

2) Count the number of customers active in each subsequent period.

3) Convert the counts to percents

4) Count the number of users per cohort. Add column to table.

Our Example Cont.

```
RECALL:
CREATE TABLE cohort_totals AS
SELECT cohort_date, COUNT(DISTINCT shopper_id) AS cohort_total
FROM cohort_assignment
GROUP BY cohort_date;
```

So the extra rows we want are the same as ir olumn...

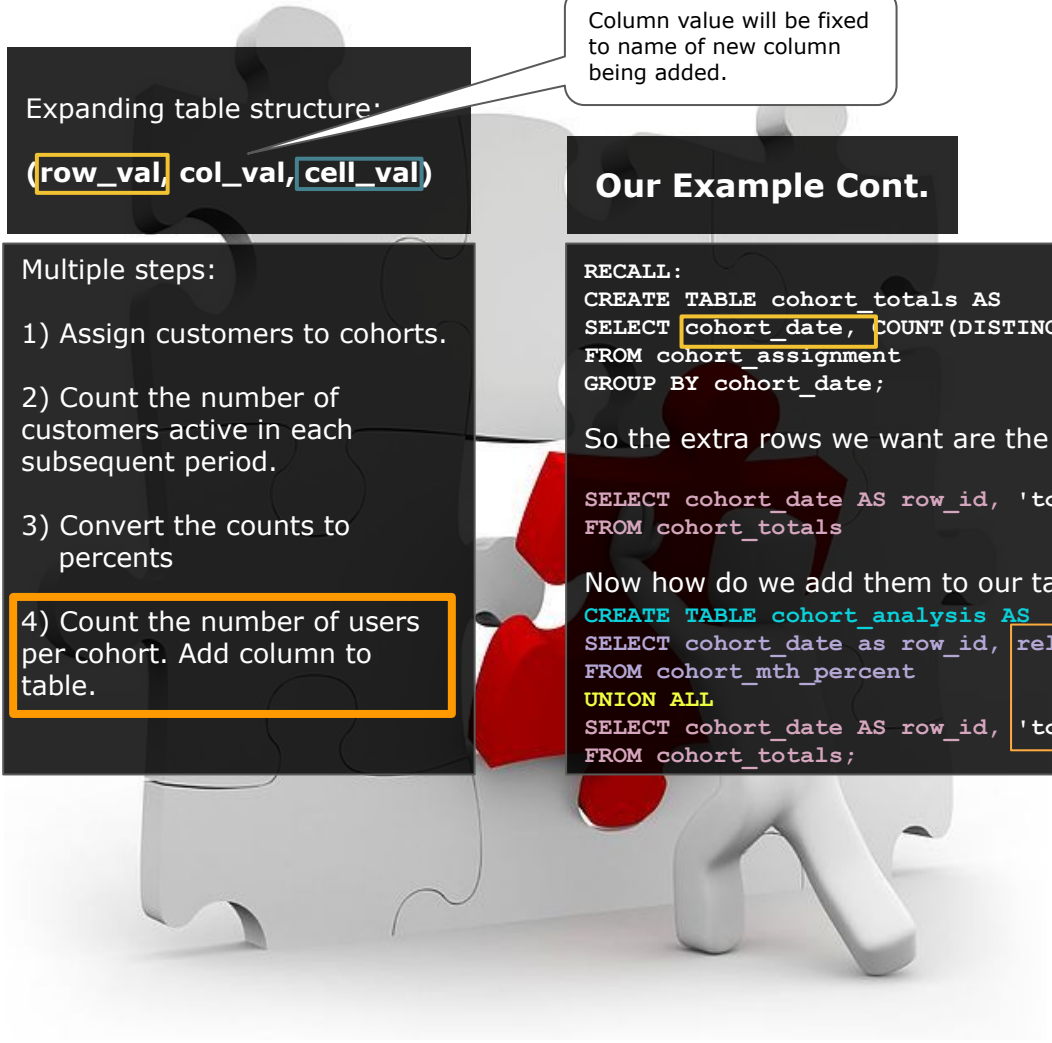
```
SELECT cohort_date AS row_id, 'total' AS co
FROM cohort_totals
```

Now how do we add them to our table cohort_mth_ ent (represented in tuple format)?

```
CREATE TABLE cohort_analysis AS
SELECT cohort_date as row_id, relative_period::STRING as col_id, active_percent as val
FROM cohort_mth_percent
UNION ALL
SELECT cohort_date AS row_id, 'total'::STRING AS col_id, cohort_total AS val
FROM cohort_totals;
```

UNION requires all column types to be the same **in our relation.**

In our **expanding table** our column header values used to be integers. Now we want to name a column header 'total'. **All must be STRING.**



Example 7

Your turn 9

Expanding table structure:

(**row_val**, **col_val**, **cell_val**)

Column value will be fixed to name of new column being added.

Multiple steps:

1) Assign customers to cohorts.

2) Count the number of customers active in each subsequent period.

3) Convert the counts to percents

4) Count the number of users per cohort. Add column to table.

Our Example Cont.

```
RECALL:
CREATE TABLE cohort_totals AS
SELECT cohort_date, COUNT(DISTINCT shopper_id) AS cohort_total
FROM cohort_assignment
GROUP BY cohort_date;
```

So the extra rows we want are the same as ir… olumn...

```
SELECT cohort_date AS row_id, 'total' AS co
FROM cohort_totals
```

UNION requires all column types to be the same **in our relation.**

In our **expanding table** our column header values used to be integers. Now we want to name a column header 'total'. **All must be STRING.**

Now how do we add them to our table cohort_mth_… (represented in tuple format)?

```
CREATE TABLE cohort_analysis AS
SELECT cohort_date as row_id, relative_period::STRING as col_id, active_percent as val
FROM cohort_mth_percent
UNION ALL
SELECT cohort_date AS row_id, 'total'::STRING AS col_id, cohort_total AS val
FROM cohort_totals;
```

Let's recap.

Expanding table structure:
(row_val, col_val, cell_val)

Multiple steps:

1) Assign customers to cohorts.

2) Count the number of customers active in each subsequent period.

3) Convert the counts to percents

4) Count the number of users per cohort. Add column to table.

```
CREATE TABLE cohort_assignment
AS
SELECT shopper_id,
       DATE_TRUNC('month',
MIN(purchased_at))::DATE as
cohort_date
FROM transactions
GROUP BY shopper_id;
```

```
CREATE TABLE cohort_mth_cts AS
SELECT
  cohort_date,
  MONTHS_BETWEEN(
    DATE_TRUNC('month',
                purchased_at),
    DATE_TRUNC('month',
                cohort_date)
  ) as relative_period,
  COUNT(DISTINCT shopper_id)
                 as active_ct
FROM transactions
JOIN cohort_assignment
USING (shopper_id)
GROUP BY cohort_date,
relative_period;
```

```
CREATE TABLE cohort_totals AS
SELECT cohort_date,
COUNT(DISTINCT shopper_id) AS
cohort_total
FROM cohort_assignment
GROUP BY cohort_date;


CREATE TABLE
cohort_mth_percent AS
SELECT cohort_date,
relative_period, active_ct /
cohort_total as active_percent
FROM cohort_mth_cts
JOIN cohort_totals
USING (cohort_date)
```

```
CREATE TABLE cohort_analysis
AS
SELECT cohort_date as row_id,
relative_period::STRING as
col_id, active_percent as val
FROM cohort_mth_percent
UNION ALL
SELECT cohort_date AS row_id,
'total'::STRING AS col_id,
cohort_total AS val
FROM cohort_totals
```

Let's recap.

Expanding table structure:
(row_val, col_val, cell_val)

Multiple steps:

1) Assign customers to cohorts.

2) Count the number of customers active in each subsequent period.

3) Convert the counts to percents

4) Count the number of users per cohort. Add column to table.

```
CREATE TABLE cohort_assignment
AS
SELECT shopper_id,
       DATE_TRUNC('month',
MIN(purchased_at))::DATE as
cohort_date
FROM transactions
GROUP BY shopper_id;
```

```
CREATE TABLE cohort_mth_cts AS
SELECT
  cohort_date,
  MONTHS_BETWEEN(
    DATE_TRUNC('month',
                purchased_at),
    DATE_TRUNC('month',
                cohort_date)
  ) as relative_period,
  COUNT(DISTINCT shopper_id)
                 as active_ct
FROM transactions
JOIN cohort_assignment
USING (shopper_id)
GROUP BY cohort_date,
relative_period;
```

```
CREATE TABLE cohort_totals AS
SELECT cohort_date,
COUNT(DISTINCT shopper_id) AS
cohort_total
FROM cohort_assignment
GROUP BY cohort_date;


CREATE TABLE
cohort_mth_percent AS
SELECT cohort_date,
relative_period, active_ct /
cohort_total as active_percent
FROM cohort_mth_cts
JOIN cohort_totals
USING (cohort_date)
```

```
CREATE TABLE cohort_analysis
AS
SELECT cohort_date as row_id,
relative_period::STRING as
col_id, active_percent as val
FROM cohort_mth_percent
UNION ALL
SELECT cohort_date AS row_id,
'total'::STRING AS col_id,
cohort_total AS val
FROM cohort_totals
```

Well, that was a lot of temporary tables.

So much for auto-updating (expanding tables)....

Let's recap.

Expanding table structure:
(row_val, col_val, cell_val)

Multiple steps:

1) Assign customers to cohorts.

2) Count the number of customers active in each subsequent period.

3) Convert the counts to percents

4) Count the number of users per cohort. Add column to table.

```
CREATE TABLE cohort_assignment
AS
SELECT shopper_id,
       DATE_TRUNC('month',
MIN(purchased_at))::DATE as
cohort_date
FROM transactions
GROUP BY shopper_id;
```

```
CREATE TABLE cohort_mth_cts AS
SELECT
  cohort_date,
  MONTHS_BETWEEN(
    DATE_TRUNC('month',
                purchased_at),
    DATE_TRUNC('month',
                cohort_date)
  ) as relative_period,
  COUNT(DISTINCT shopper_id)
                 as active_ct
FROM transactions
JOIN cohort_assignment
USING (shopper_id)
GROUP BY cohort_date,
relative_period;
```

```
CREATE TABLE cohort_totals AS
SELECT cohort_date,
COUNT(DISTINCT shopper_id) AS
cohort_total
FROM cohort_assignment
GROUP BY cohort_date;


CREATE TABLE
cohort_mth_percent AS
SELECT cohort_date,
relative_period, active_ct /
cohort_total as active_percent
FROM cohort_mth_cts
JOIN cohort_totals
USING (cohort_date)
```

```
CREATE TABLE cohort_analysis
AS
SELECT cohort_date as row_id,
relative_period::STRING as
col_id, active_percent as val
FROM cohort_mth_percent
UNION ALL
SELECT cohort_date AS row_id,
'total'::STRING AS col_id,
cohort_total AS val
FROM cohort_totals
```

Solution A: Subqueries.
→ Pretty unreadable
→ `cohort_totals` is used twice. Will have to repeat code.

Let's recap.

Expanding table structure:
(row_val, col_val, cell_val)

Multiple steps:

1) Assign customers to cohorts.

2) Count the number of customers active in each subsequent period.

3) Convert the counts to percents

4) Count the number of users per cohort. Add column to table.

```
CREATE TABLE cohort_assignment
AS
SELECT shopper_id,
       DATE_TRUNC('month',
MIN(purchased_at))::DATE as
cohort_date
FROM transactions
GROUP BY shopper_id;
```

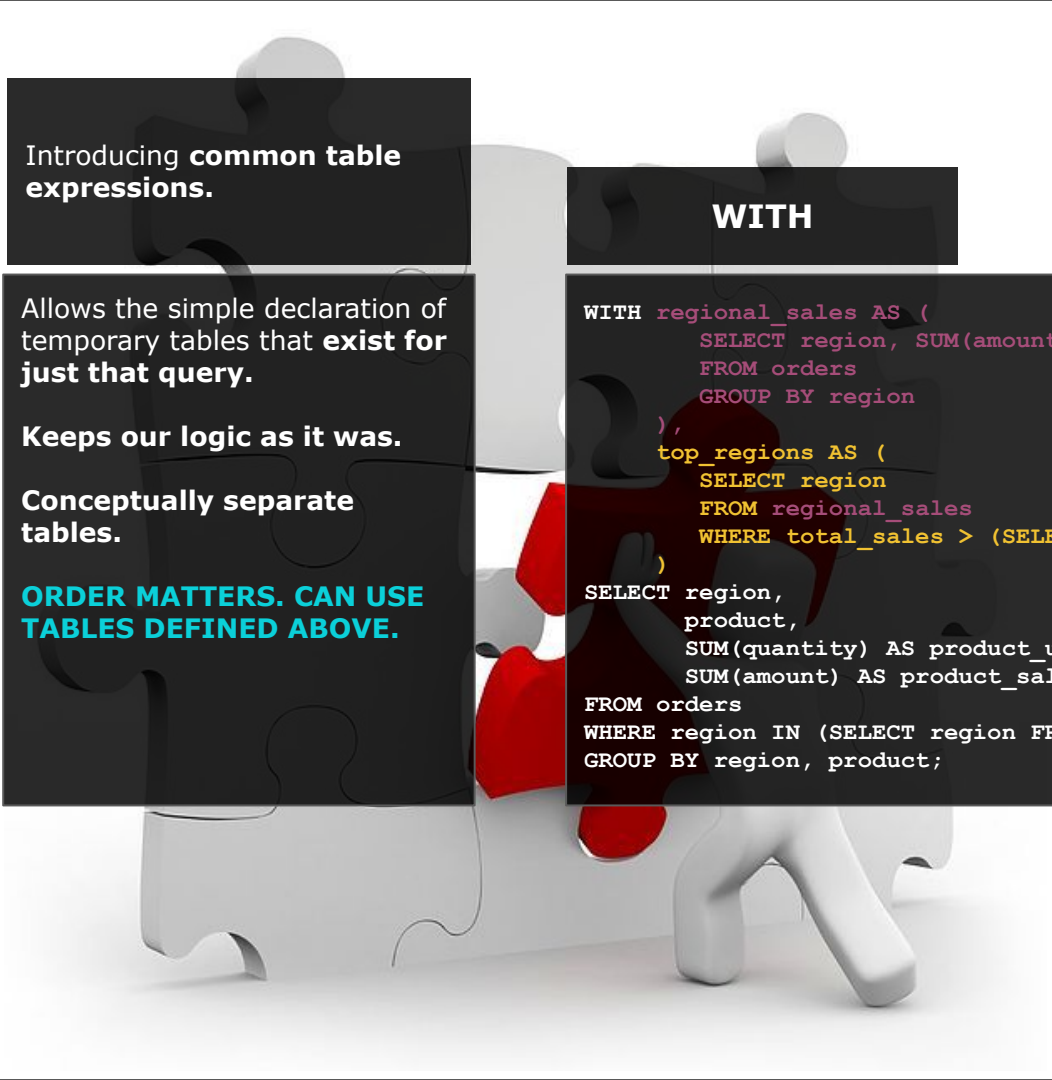
```
CREATE TABLE cohort_mth_cts AS
SELECT
  cohort_date,
  MONTHS_BETWEEN(
    DATE_TRUNC('month',
                purchased_at),
    DATE_TRUNC('month',
                cohort_date)
  ) as relative_period,
  COUNT(DISTINCT shopper_id)
                 as active_ct
FROM transactions
JOIN cohort_assignment
USING (shopper_id)
GROUP BY cohort_date,
relative_period;
```

```
CREATE TABLE cohort_totals AS
SELECT cohort_date,
COUNT(DISTINCT shopper_id) AS
cohort_total
FROM cohort_assignment
GROUP BY cohort_date;


CREATE TABLE
cohort_mth_percent AS
SELECT cohort_date,
relative_period, active_ct /
cohort_total as active_percent
FROM cohort_mth_cts
JOIN cohort_totals
USING (cohort_date)
```

```
CREATE TABLE cohort_analysis
AS
SELECT cohort_date as row_id,
relative_period::STRING as
col_id, active_percent as val
FROM cohort_mth_percent
UNION ALL
SELECT cohort_date AS row_id,
'total'::STRING AS col_id,
cohort_total AS val
FROM cohort_totals
```

Solution B: Common Table Expressions.
→ Very useful.
→ Simple.
→ **Let's learn them now!**



Introducing **common table expressions.**

WITH

Allows the simple declaration of temporary tables that **exist for just that query.**

Keeps our logic as it was.

Conceptually separate tables.

ORDER MATTERS. CAN USE TABLES DEFINED ABOVE.

```
WITH regional_sales AS (
        SELECT region, SUM(amount) AS total_sales
        FROM orders
        GROUP BY region
    ),
    top_regions AS (
        SELECT region
        FROM regional_sales
        WHERE total_sales > (SELECT SUM(total_sales)/10 FROM regional_sales)
    )
SELECT region,
       product,
       SUM(quantity) AS product_units,
       SUM(amount) AS product_sales
FROM orders
WHERE region IN (SELECT region FROM top_regions)
GROUP BY region, product;
```

Let's recap.

Expanding table structure:
(row_val, col_val, cell_val)

Multiple steps:

1) Assign customers to cohorts.

2) Count the number of customer's active each in each subsequent period.

3) Convert the counts to percents

4) Count the number of users per cohort. Add column to table.

```
CREATE TABLE cohort_assignment
AS
SELECT shopper_id,
       DATE_TRUNC('month',
MIN(purchased_at))::DATE as
cohort_date
FROM transactions
GROUP BY shopper_id;
```

```
CREATE TABLE cohort_mth_cts AS
SELECT
  cohort_date,
  MONTHS_BETWEEN(
    DATE_TRUNC('month',
                purchased_at),
    DATE_TRUNC('month',
                cohort_date)
  ) as relative_period,
  COUNT(DISTINCT shopper_id)
                 as active_ct
FROM transactions
JOIN cohort_assignment
USING (shopper_id)
GROUP BY cohort_date,
relative_period;
```

```
CREATE TABLE cohort_totals AS
SELECT cohort_date,
COUNT(DISTINCT shopper_id) AS
cohort_total
FROM cohort_assignment
GROUP BY cohort_date;


CREATE TABLE
cohort_mth_percent AS
SELECT cohort_date,
relative_period, active_ct /
cohort_total as active_percent
FROM cohort_mth_cts
JOIN cohort_totals
USING (cohort_date)
```

Example 7

Your turn 10

Using CTEs write the whole thing as one SQL query.

```
CREATE TABLE cohort_analysis
AS
SELECT cohort_date as row_id,
relative_period::STRING as
col_id, active_percent as val
FROM cohort_mth_percent
UNION ALL
SELECT cohort_date AS row_id,
'total'::STRING AS col_id,
cohort_total AS val
FROM cohort_totals
```

Common Table Expression Syntax Example

```
WITH regional_sales AS (
        SELECT … FROM … WHERE …
    ),
    top_regions AS (
        SELECT … FROM …
    )
SELECT … ;
```

```sql
CREATE TABLE cohort_analytis_final AS
WITH    cohort_assignment AS (
            SELECT shopper_id,
            DATE_TRUNC('month', MIN(purchased_at))::DATE as cohort_date
            FROM transactions
            GROUP BY shopper_id
        ),
        cohort_mth_cts AS (
            SELECT cohort_date,
                MONTHS_BETWEEN( DATE_TRUNC('month',purchased_at), DATE_TRUNC('month', cohort_date) )
                   as relative_period,
                COUNT(DISTINCT shopper_id) as active_ct
            FROM transactions
            JOIN cohort_assignment
            USING (shopper_id)
            GROUP BY cohort_date, relative_period
        ),
        cohort_totals AS (
            SELECT cohort_date, COUNT(DISTINCT shopper_id) AS cohort_total
            FROM cohort_assignment
            GROUP BY cohort_date
        ),
        cohort_mth_percent AS (
            SELECT cohort_date, relative_period, active_ct / cohort_total as active_percent
            FROM cohort_mth_cts
            JOIN cohort_totals
            USING (cohort_date)
        )
SELECT cohort_date as row_id, relative_period::STRING as col_id, active_percent as val
FROM cohort_mth_percent
UNION ALL
SELECT cohort_date AS row_id, 'total'::STRING AS col_id, cohort_total AS val
FROM cohort_totals;
```

Yep. Simple. But complicated.

Just takes practice and step-by-step logical thought.

What we've done this week:

→ 7 Examples of KPIs in SQL

→ Cohort Analysis

→ UNION

→ Expanding tables

→ Visualizing expanding tables in Tableau