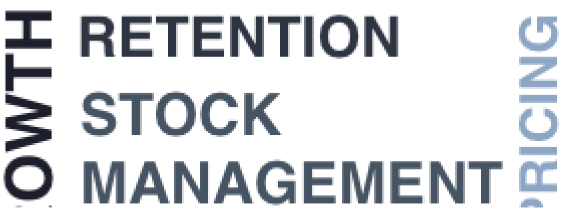
E-commerce Analysis: Data-Structures and Applications

Julien Kervizic Follow
Dec 25, 2018 · 11 min read ★

Having effective data structures, enables and empowers different types of analyses to be performed. Overall looking at some of the traditional use cases for e-commerce, we can extract 5 different distinct areas of focus that we might want to address:



We could also add to this data structures related to clickstreams and conversion optimization, marketing spend, product recommendations … but these would need to be the subject of a separate post.

RETENTION STUDIES

Retention studies aim to provide a better understanding which customers stick or depart from the service. They try to find hypotheses as to why customer churn from the platform, hypotheses for which potential treatments can then be tested in live condition.

At the heart of retention studies lies the concept of cohort, typically defined as a group of customer having done their first action at a given point in time. For instance all the customer having made a purchase on your platform on January 2018. We must therefore address cohort modeling as part of the data-structures to be considered.The cohort modeling take as its' base a table at user level that contains an acquisition date for each customer, for instance:

CustomerId	AcquisitionDate	Name	LastName
1234	2018-01-01	Waldo	Smith
1236	2017-05-15	Luigi	Russo

This acquisition date can be based on any activity metrics that you want to consider, be it a sign-up activity, a purchase activity, a website view, … It is

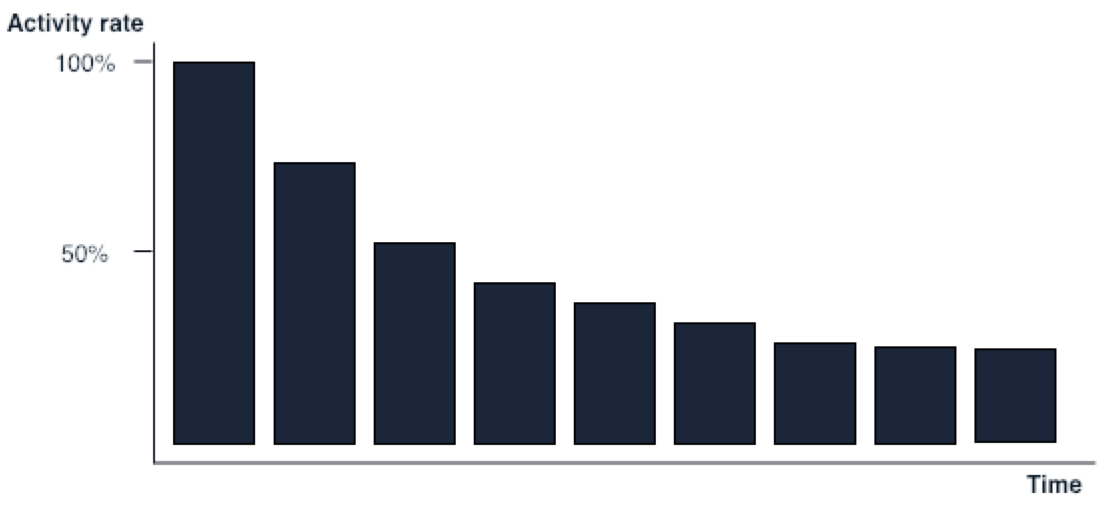
As such this table, lets call it can be populated from any activity table based on an incremental daily load for instance:

```
UPSERT INTO dim_customer
SELECT
 COALESCE(dc.CustomerId, ac.CustomerId) AS CustomerId,
 LEAST(dc.AcquisitionDate, ac.ActivityDate) AS AcquisitionDate
FROM dim_customers dc
FULL OUTER JOIN (
 SELECT
```

```sql
        CustomerId,
        ActivityDate
    FROM daily_activity
    WHERE
        ActivityType = 'signup'
        AND ActivityDate = '<RUN_DATE>'
    GROUP BY 1, 2
  ) ac
    ON dc.CustomerId = ac.CustomerId
GROUP BY 1, 2
```

dim_customer.sql hosted with ❤ by GitHub view raw

As shown above an aggregate event activity is useful for the continuous computation of the dim_customer table.For these metrics it is usually customary to have the event type being tied to the acquistionDate. This enables the nice property that all event activity will be after the acquisition date.



decay histogram

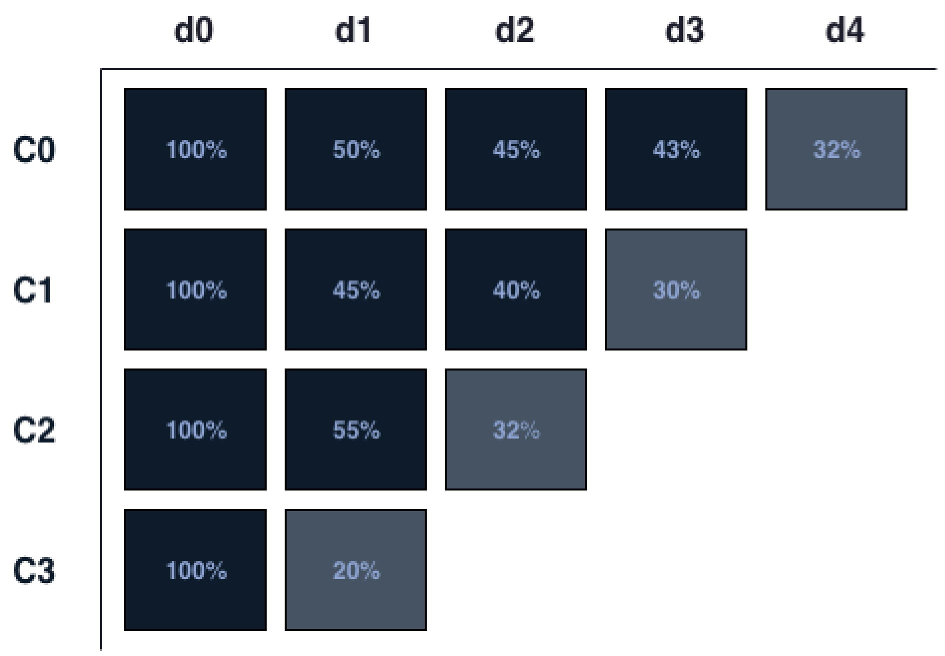
For instance, let's say that we wanted to compute a decay curve/histogram such as the one depicted on the right. The decay histogram shows the % of a user-base that is active at any given time since acquisition.

Let's assume that we have built the dim_customer table as shown above based on signup activity and that a signup is the first ever event of a sign-in event.

```sql
SELECT
        DATEDIFF(ac.ActivityDate, dc.AcquisitionDate) AS DaySinceAcquisition,
        COUNT(DISTINCT ac.CustomerId) AS D1ActiveCustomers
FROM dim_customer dc
LEFT OUTER JOIN (
        SELECT
```

```sql
                CustomerId,
                ActivityDate
        FROM agg_daily_activity
        WHERE
                ActivityType IN ('signup', 'signin')
                AND ActivityDate >= '<MIN_DATE>'
        GROUP BY 1,2
) ac
        ON dc.CustomerId = ac.CustomerId
        AND dc.AcquisitionDate <= ac.ActivityDate
WHERE
        dc.AcquisitionDate => '<MIN_DATE>'
GROUP BY 1
```

The above SQL Query for instance enables to compute the datasets needed for this purpose. Since we know that everybody within the user base is active at their Acquisition date (DaySinceAcquisition = 0), we only have to represent D1ActiveCustomers as a proportion of user active at DaySinceAcquisition = 0 to compute the decay histogram.

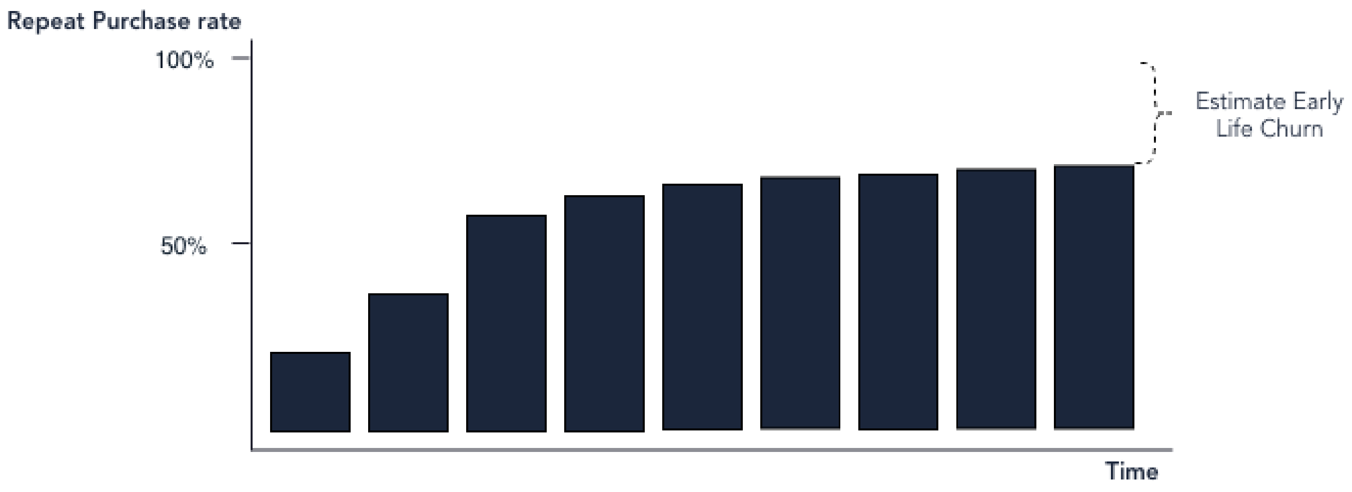


Triangle charts are an evolution of decay curves/histogram, introducing the concept of cohort to the mix. Since we already are computing the acquisition date in order to compute a decay curve, in order to output a triangle chart, the only thing we have to do is to surface that information:

```sql
SELECT
        dc.AcquisitionDate,
        DATEDIFF(ac.ActivityDate, dc.AcquisitionDate) AS DaySinceAcquisition,
        COUNT(DISTINCT ac.CustomerId) AS D1ActiveCustomers
        COUNT(DISTINCT dc.CustomerId) AS D1ActiveCustomers
FROM dim_customer dc
LEFT OUTER JOIN (
        SELECT
                CustomerId,
                ActivityDate
        FROM agg_daily_activity
        WHERE
                ActivityType IN ('signup', 'signin')
                AND ActivityDate >= '<MIN_DATE>'
        GROUP BY 1,2
) ac
        ON dc.CustomerId = ac.CustomerId
        AND dc.AcquisitionDate <= ac.ActivityDate
WHERE
        dc.AcquisitionDate >= '<MIN_DATE>'
GROUP BY 1,2
```

cohort_active_customer.sql hosted with ❤ by GitHub view raw

Having other type of attributes within it enables the computation of the datasets needed for repeat purchase rates tables.



In order to compute the repeat purchase we need to calculate the number of days it took for a given user to achieve his/her second purchase. In this case we still need to keep the concept of cohort in place but we need to augment it with the concept of quantity purchased.

```sql
SELECT
        dc.CustomerId,
        dc.AcquisitionDate,
        DATEDIFF(ac.ActivityDate, dc.AcquisitionDate) AS DaySinceAcquisition,
        SUM(ac.D1Purchases) A D1Purchases
FROM dim_cust_purchaser dc
```

```
LEFT OUTER JOIN (
        SELECT
                CustomerId,
                ActivityDate,
                SUM(D1Quantity) AS D1Purchases
        FROM agg_daily_activity
        WHERE
                ActivityType IN ('purchase')
                AND ActivityDate >= '<MIN_DATE>'
        GROUP BY 1,2
) ac
        ON dc.CustomerId = ac.CustomerId
        AND dc.AcquisitionDate <= ac.ActivityDate
WHERE
        dc.AcquisitionDate >= '<MIN_DATE>'
GROUP BY 1,2,3
```

uid_cohort_metrics.sql hosted with ❤ by GitHub view raw

In order to compute the repeat purchase rate, we must first compute for each customer, the number of purchases made with respect to its acquisition date. In a second step we must compute the number of days it took to achieve 2 Purchases. For each customer, we must therefore compute the rollup of all their daily purchases in order to obtain that estimate. Since daily purchases can be numerous (>1), we must further add logic to check that the previous day had less than 2 rolled purchases to identity the date as the repurchase date.

Now let's assume that we are computing for each day the query: uid_cohort_metrics.sql and providing the result into a table called daily cohort activity. We run the query every day with a filter on activity date that we export as a column:

```
SELECT
        dc.CustomerId,
        dc.AcquisitionDate,
        ac.ActivityDate,
        DATEDIFF(ac.ActivityDate, dc.AcquisitionDate) AS DaySinceAcquisition,
        SUM(ac.D1Purchases) A D1Purchases
FROM dim_cust_purchaser dc
LEFT OUTER JOIN (
        SELECT
                CustomerId,
                ActivityDate,
                SUM(D1Quantity) AS D1Purchases
        FROM agg_daily_activity
        WHERE
                ActivityType IN ('purchase')
```

```sql
                AND ActivityDate >= '<MIN_DATE>'
                AND ActivityDate = '<RUN_DATE>'
        GROUP BY 1,2
) ac
        ON dc.CustomerId = ac.CustomerId
        AND dc.AcquisitionDate <= ac.ActivityDate
WHERE
        dc.AcquisitionDate >= '<MIN_DATE>'
GROUP BY 1,2,3,4
```

Based on this table we can compute a daily rollup cohort activity:

```sql
SELECT
        COALESCE(dca.CustomerId, lca.CustomerId) AS CustomerId,
        COALESCE(dca.AcquisitionDate, lca.AcquisitionDate) AS AcquisitionDate,
        '<RUN_DATE>' AS ActivityDate ,
        DATEDIFF('<RUN_DATE>', COALESCE(dca.AcquisitionDate, lca.AcquisitionDate)) AS Da
        COALESCE(dca.D1Purchases,0) A D1Purchases,
        COALESCE(lca.D1Purchases, 0) + COALESCE(lca.DxPurchases,0) AS DxPurchases
FROM (SELECT * FROM daily_cohort_activity WHERE ActivityDate = '<RUN_DATE>') dca
FULL OUTER JOIN lifetime_cohort_activity lca
        ON dca.CustomerId = lca.CustomerId
        AND lca.ActivityDate = '<RUN_DATE-1>'
```

After which, we can finally compute effectively the re-purchase rate:

```sql
SELECT
        DaysToRepurchase,
        COUNT(DISTINCT CustomerId) as Customers
FROM (
        SELECT
                CustomerId,
                MIN(DaySinceAcquisition) as DaysToRepurchase
        FROM rollup_daily_cohort_activity
        WHERE DxPurchases > 1
        GROUP BY 1
) rp
GROUP BY 1
```

The above query giving us the time and count of customers having turned into a repurchaser after N given days since acquisition. In order to compute the repurchase rate graph, all that is left to do once this dataset is obtained it to make it a running sum and normalize it to the population size.

Setting up a few data-structures based on the above queries significantly helps to get a better understanding how retention behavior across your platform. The most important to setup are a customer level table that contains attributes of the customer both static and computed, customer event aggregation tables and customer cohort level tables. Having these setup makes it fairly easy to retrieve the information necessary in order to compute the different retention graphs and charts, from decay curves, triangle charts to repeat purchase histograms.

ENGAGEMENT STUDIES

Different framework for understanding customer engagement exists out there. They exist to understand how different segment of the customer base interact with your platform. Two of the most frequent frameworks used to get a sense of engagement are RFM and power user curves.

RFM is a segmentation methodology that works through the computation of 3 specific dimension, recency, frequency and monetary value. The RFM then place a score for each customer across each of these dimensions, with the higher values representing the most desirable outcome.

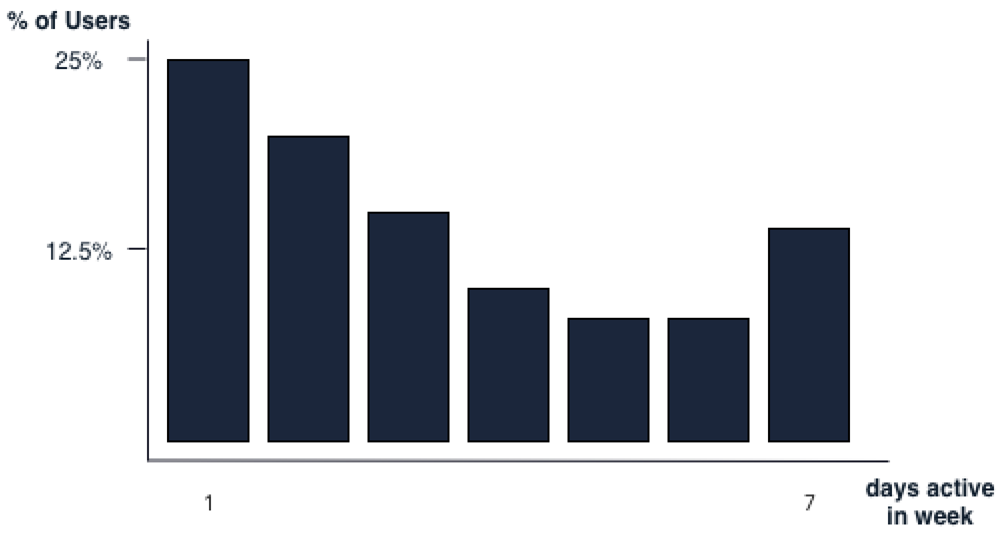
The first dimension, recency can be obtained by a slight modification to the dim_purchaser table and query to include a last purchase date.

```sql
UPSERT INTO dim_cust_purchaser
SELECT
 COALESCE(dc.CustomerId, ac.CustomerId) AS CustomerId,
 LEAST(dc.AcquisitionDate, ac.ActivityDate) AS AcquisitionDate,
 GREATEST(dc.LastPurchaseDate, ac.ActivityDate) AS LastPurchaseDate
FROM dim_cust_purchaser dc
FULL OUTER JOIN (
 SELECT
  CustomerId,
  ActivityDate
 FROM daily_activity
 WHERE
  ActivityType = 'purchase'
  AND ActivityDate = '<RUN_DATE>'
 GROUP BY 1, 2
)
```

```
) ac
 ON dc.CustomerId = ac.CustomerId
GROUP BY 1, 2
```

dim_cust_purchaser.sql hosted with ❤ by GitHub view raw

The frequency and monetary value metrics are normally taken as being trailing metrics such as trailing measurements such as trailing twelve months (TTM). These calculation can be made by enhancing the daily rollup cohort activity query with a value measure and the necessary trailing metrics.



Power User curves are a different mean to get some information as to how your customer base is trending in terms of activity. It relies on a metrics of days of activity, Facebook had defined an *L30* measure to define the total number of days, users were active in a given month. The shape and trend of the power user curves gives a sense of user activity and engagement within the time period, allowing the identification of power users across the customer base. An *L28* measure is sometimes used instead of an *L30* measure as a proxy for the monthly activity, due to the computational advantages of *L28*. *L28* can be computed as the sum of four *L7* weekly measures, and l7 measures themselves can be computed as the sum of 7 L1.

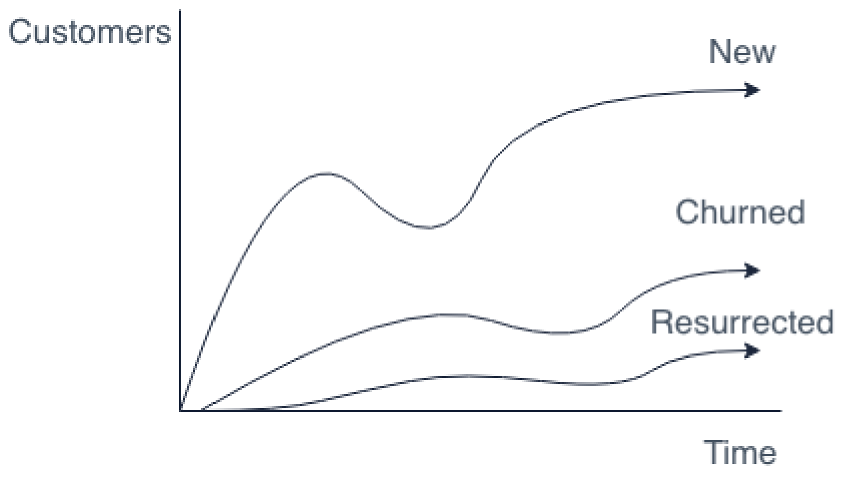
```
SELECT
        CustomerId,
        SUM(CASE
            WHEN D1Purchases > 0 and ActivityDate = '<RUN_DATE>' THEN 1
            ELSE 0 END) AS a l1,
        SUM(CASE WHEN D1Purchases > 0 THEN 1 ELSE 0 END) AS a l7
FROM daily_rollup_cohort_activity
WHERE ActivityDate BETWEEN '<RUN_DATE-6>' AND '<RUN_DATE>'
GROUP BY 1
```

l7.sql hosted with ❤ by GitHub view raw

Engagement studies only need little tweaks on the data-structures needed for retention studies in order to be catered for. At their core, they rely on information being available at customer level with trailing measures, and these measures be grafted on those needed for retention studies with ease.

GROWTH ACCOUNTING

Growth Accounting attempts to split your customer base into different lifestage status and understand the evolution of your active customer base through these statuses. For this purpose A customer can be in a stage of a new customer, a resurrected, retained, churned or potentially stale.



Growth Accounting can help you understand if you are effectively retaining or resurrecting your customer for effective growth or if you are facing a headwind with respect to growth due to churn.

The definition of Active User/Customers within growth accounting is primordial, it usually refers to users having performed a given action in the past X days.

The following two equations define how growth accounting is split out across lifestage:

$$\text{Active}(t) = \text{New}(t) + \text{Resurrected}(t) + \text{Retained}(t)$$

$$\text{Retained}(t) = \text{Active}(t\text{-}1) — \text{Churned}(t)$$

For a more thorough explanation of growth accounting:

Growth Accounting & Triangle Heatmap Explanation

Facebook for Developers • Follow

Share

More Videos on Facebook Watch

1:51

4:25

27:47

Facebook for Developers

Facebook for Developers

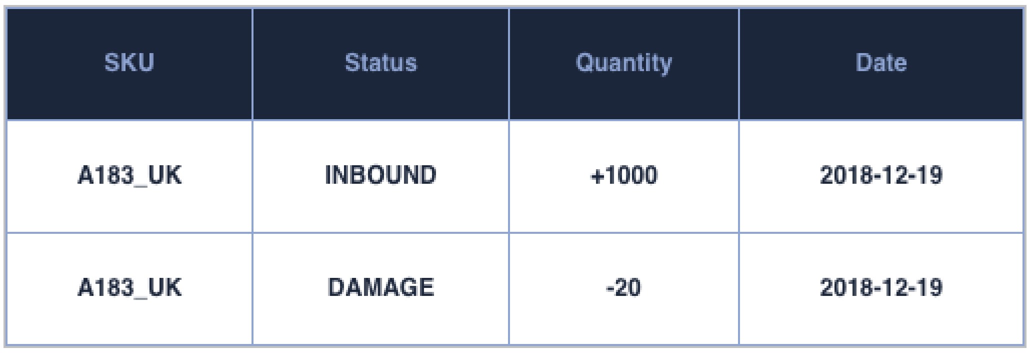
Facebook Presents #TechbyHer Panel Discusion

Facebook for Developers

-6:43 HD

Here's a clear explanation of the Growth charts in App Insights. The Growth Accounting chart helps you determine exactly what is driving user growth by separating installs, uninstalls, stale users, revived users, and retained users. The Triangle Heatmap chart is available for apps with more than 25,000 monthly active users and shows how different cohorts of new users remain engaged over time and patterns that indicate what has affected engagement. These charts can be accessed through App Insights under Installs > Growth.

562 **168** **136**



SKU	Status	Quantity	Date
A183_UK	INBOUND	+1000	2018-12-19
A183_UK	DAMAGE	-20	2018-12-19

In essence growth accounting assign to a user a status based on its' activity today and his activity status of yesterday. Let's say we have two tables, one containing the daily *Lx* values for customers the other meant to store the growth accounting status of the customers, then we could compute the new growth accounting status like this:

```
SELECT
        '<RUN_DATE>'
        COALESCE(a.CustomerId, b.CustomerId) AS CustomerId,
        CASE
                WHEN (b.growthacc_status is null
                        AND a.L7 > 0) THEN 'New'
                WHEN (b.growthacc_status IN ('New', 'Resurrected', 'Retained')
                        AND a.L7 > 0) THEN 'Retained'
                        AND a.L7 > 0) THEN 'Retained'
                WHEN (b.growthacc_status IN ('Churned', 'Stale')
                        AND a.L7 > 0) THEN 'Resurrected'
                WHEN (b.growthacc_status IN ('New', 'Resurrected', 'Retained')
                        AND a.L7 = 0) THEN 'Churned'
                WHEN *b.growthacc_status IN ('Churned')
                        AND a.L7 = 0) Then 'Stale'
                ELSE 'Unknown'
        END AS growthacc_status

FROM (SELECT * FROM customer_l7_status WHERE ActivityDate = '<RUN_DATE>') a
FULL OUTER JOIN custmer_growth_accounting_status b
        ON b.CustomerId = a.CustomerId
        AND b.ds = DATE_ADD(a.ds, INTERVAL -1 DAY)
```

growthacc.sql hosted with ❤ by GitHub   view raw

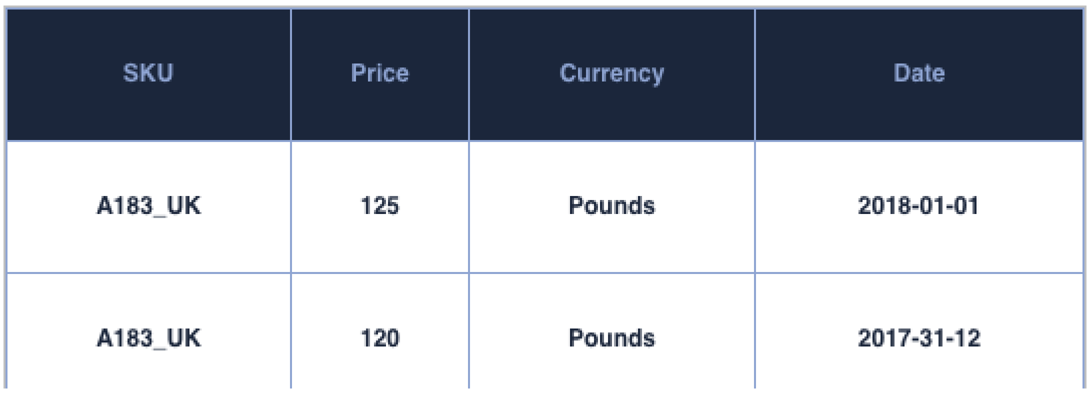
The status column for growth accounting could be potentially added to any date / customer table already existing to provide that information. For instance if dim_customers contained a snapshot of all customers present on a given date the growth accounting status could potentially be placed there.

PRICING STUDIES

There is a wide range of pricing studies that are of interest on a website, from study of change in Price Discount Perception (Sales Price / Recommended Retail Price), Price Elasticities, revenue cannibalization …

For pricing studies it is important to have a snapshot table or a type II history table that contains the pricing history of each item available on the website.

A snapshot table would contain the state of the different prices active on a given day / hour etc.. If price would change within that interval only one of the value would be captured. Nevertheless the use of a snapshot table in order to study pricing behavior can be quite effective. It allows for simple query to be run and see how price has impacted purchasing behavior for instance.



SKU	Price	Currency	Date
A183_UK	125	Pounds	2018-01-01
A183_UK	120	Pounds	2017-31-12

Depending on the source system used, it might be difficult to understand when a price has been change and have a certain history or pricing changes. In these cases a snapshot table is recommended, where the data feeding into it would be obtained at certain recurring intervals. Simple queries can be run on these type of tables to understand for instance when prices have been changed:

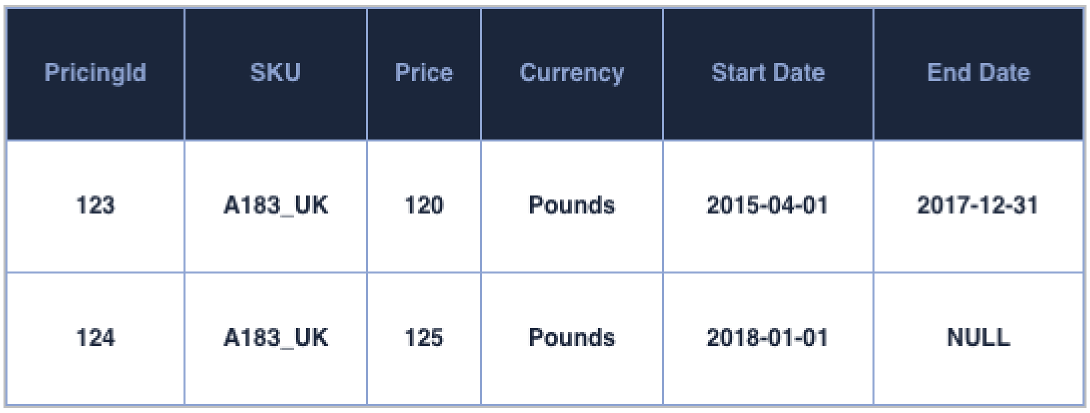
```sql
SELECT
        'price_change' AS event_name
        a.date,
        a.sku,
        a.price AS new_price,
        b.price AS old_price
FROM snapshot_catalogue_prices a
INNER JOIN snapshot_catalogue_prices b
        ON b.date = DATE_ADD(a.date, INTERVAL -1 DAY)
        AND b.sku = a.sku
        AND b.price != a.price
```

As well as for instance getting a quick understanding of the discount level active on the site:

```sql
SELECT
        a.date,
        a.sku,
        SUM(b.price*a.quantity) AS gmv,
        SUM((b.price - a.item_price) * a.quantity) AS total_discount_amount
FROM o_order_item a
INNER JOIN snapshot_catalogue_prices b
        ON a.sku = b.sku
        AND a.date = b.date
WHERE a.date BETWEEN '<RUN_DATE-X>' AND '<RUN_DATE>'
GROUP BY 1,2
```

Price snapshots tables makes these kind of queries fairly easy to compute.

Another data structure that is useful for pricing studies is history or type II tables. Type II tables hold pricing information that is valid for a certain range of time as shown in the table below:



PricingId	SKU	Price	Currency	Start Date	End Date
123	A183_UK	120	Pounds	2015-04-01	2017-12-31
124	A183_UK	125	Pounds	2018-01-01	NULL

Since type II table holds a range, they can handle without any issues granular switchover, for instance they could host a start_time in milliseconds to provide the information as to how long the given price was active. Beside this, the main advantage of this data structure compared to a price snapshot is the significantly lower size of the table when price changes occurs at a low-medium frequency.

Type II history table can for instance be particularly useful for pricing studies to understand how much we should per day based on each pricing period:

```sql
SELECT
        b.pricingId,
        b.start_date,
        b.sku,
        b.price,
        b.end_date - b.start_date AS pricing_duration,
        SUM(a.quantity) AS quantity
FROM o_order_item a
INNER JOIN history_catalogue_prices b
        ON a.date BETWEEN b.start_date and b.end_date
        AND a.sku = b.sku
WHERE
        a.date BETWEEN '<RUN_DATE-X>' AND '<RUN_DATE>'
GROUP BY 1,2,3,4,5
```

Overall both snapshot and history tables help shed lights to diverse pricing studies. Having both data structures helps gain efficiencies in the analysis of pricing movements but it is however necessary to take into account the limitations of snapshot tables within these analyses.

Stock Management

Efficient inventory management can sometimes be challenging depending on the data-feed that are provided. The minimum that is needed to be able to run a web-shop only relies on two columns SKU and quantity sellable. For analytics purposes a lot more attributes is desirable.

For instance to deal with perishable products, both having visibility of lot number and expiration date would allow for visibility on the products likely to be scrapped. For goods receipt and damaged goods having a status within inventory would allow to better understand what volume of the SKU could potentially be repaired/refurbished and put back into inventory or salvaged for spare parts.

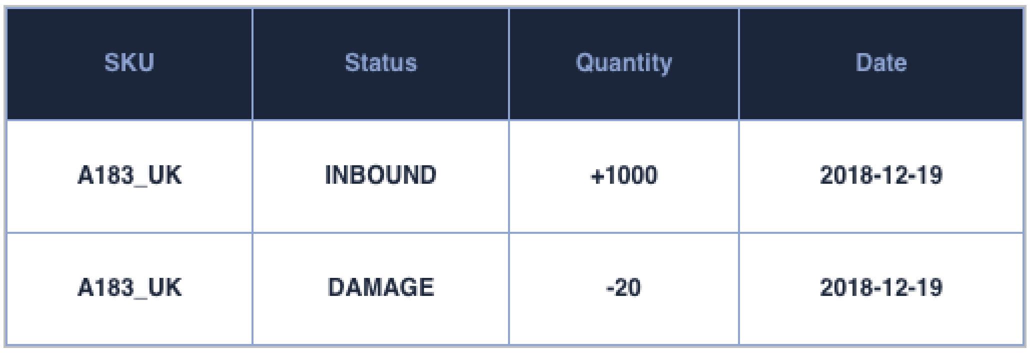
SKU	Quantity	Date
A183_UK	2002	2018-12-20
A249_UK	120	2018-12-20

One potential use of this data-structure is to calculate the number of days on hand (doh) of inventory. Days on Hand is a measure to understand if we might be over or understocked for particular items. If for instance we wanted to calculate doh based on the trailing 7 days sales using this data structure:

```sql
SELECT
        a.sku,
        a.quantity AS inventory_quantity,
        a.quantity / b.daily_quantity_sold AS doh
FROM inventory_report a
INNER JOIN (
        SELECT
                sku,
                SUM(quantity)/7 AS daily_quantity_sold
        FROM o_order_items c
        WHERE
                c.date BETWEEN '<RUN_DATE-6>' AND '<RUN_DATE>'
        GROUP BY 1
        ) b
        ON a.sku = b.sku
WHERE a.date = '<RUN_DATE>'
```

doh.sql hosted with ❤️ by **GitHub** view raw

Inventory variation is another topic in stock management, that requires its own data structures. Having visibility on the cause of inventory movement helps shed light on what happens operationally, be it damages, inventory transfers across warehouses or stock inbound. At the very minimum a data-structure should contain SKU information, status code, quantity and date of the movements:



SKU	Status	Quantity	Date
A183_UK	INBOUND	+1000	2018-12-19
A183_UK	DAMAGE	-20	2018-12-19

Having this information can allow us for instance to both reconcile our stock position as the sum of these events, but also for instance to calculate KPIs such as damage rate as well as other operational metrics.

Wrapping up

Retention, Engagement and Growth are all user focused area and it is therefore no surprise that they would require similar data-structures to be built to cater for their analysis and reporting, namely:

- customer table
- customer event aggregate
- customer cohort aggregates.

Pricing studies can in turn rely on two type of data structure to help with the analysis, essentially containing the same information, but making certain queries on it more accessible, Price Snapshots and Price History tables.

Stock management in turn also relies on two data structure an Inventory table and an inventory variation table. The inventory table is highly dependent on the number of attributes and granularity that can be obtained from source systems, while inventory variation table might be construed as a consolidation of multiple events.

E-Commerce is a vast domain that enables a wide scope of analysis and having efficient data structure such as those listed above that helps with that provide significant value.

. . .

More from me on Hacking Analytics:

- One the evolution of Data Engineering
- Principles of e-commerce Personalization
- On customer lifetime value in ecommerce
- ON the attribution and decomposition of mix and rate effects
- Hacking up a reporting pipeline using Python and Excel

Business Intelligence Data Science Data Engineering Analytics Ecommerce