# Cheat Sheat

## ARM

$$Support(s) = Count(s)/n$$

$$Support(L \rightarrow R) = Count(L \cup R)/n$$

$$Confidince(l \rightarrow R) = Count(L \cup R)/Count(L)$$

$$Lift(L \rightarrow R) = \frac{Count(L \cup R)}{Count(L) \times Count(R)}$$

$$Leverage(L \rightarrow R) = Support(L \cup R) - (Support(L) \times Support(R))$$

- Support of a itemset is the proportion of transactions in the database that contain all the items in S.
- Support of a rule is the proportion of transactions in which the items in L and R occur together.
- Confidence of a rule is the proportion of transactions for which the rule is satisfied.
- Support applies to the entire database

## Apori

```
Create L_1 = set of supported itemsets of cardinality 1
Set k to 2
while(L_{k-1} != {})
{
        Create C_k from L_{k-1}
        Prune all the itemsets in C_k that are not supported, to create C_k
        k = k +1
}
```

## Apori-gen

```
C_k  = empty;
//join step
for( A : L_{k-1})
{
   for(B : L_{k-1})
   {
      if(A!=B)
      {
         if(A.sub(0,k-2) == B.sub(0,k-2) && !C_k.contains(union(A,B)))
              C_k .add(union(A,B));
       }
    }
}
//prune step
for( c : C_k)
{
   for( sub : subset(c, k-1))
   {
     if( !L_{k-1}.includes(sub) )
     {
        C_k.remove(c);
        break;
     }
   }
}
return C_k;
```

# Classification

## TDIDT

If all instances in the training set belong to the same class.
THEN return the value of the class
ELSE

a) Select an attribute A to split on
b) Sort the instances in the training set into subsets, one for each value of A.
c) return a tree with one branch for each non-empty subset, each branch having a descendant sub-tree or a class value produced by applying the algorithm recursively.

**Information gain** is defined as the difference between the original information requirement and the new requirement. It tells us how much would be gained by branching on a attribute.
*Biased toward tests with many outcomes, prefers to select attributes having a large number of values. What if you have a attribute that is a unique identifier, result in a large number of partitions*

Steps:
First calculate the expected information needed to classify a tuple in D:

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

$$p_i = |C_{i,D}| / |D|$$

$p_i$ is the probability of that a tuple D belongs to class $C_j$

for each attribute A:
{

Calculate the expected information required to classify a tuple from D based on the partitioning by A

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j)$$

Calculate the information gain for the attribute A:

$$Gain(A) = Info(D) - Info_A(D)$$

}
Then choose the attribute with the highest information gain to split on.

**Gain ratio** attempts to over come the bias with information gain on attributes having a large number of values. It does this by applying a form of normalization to info gain using a split information value:

$$SplitInfo_A(D) = -\sum_{j=1}^{v} \frac{|D_j|}{|D|} \times \log_2\left(\frac{|D_j|}{|D|}\right)$$

SplitInfo considers the number of tuples having the outcome with respect to the total number of tuples in D.

$$GainRatio(A) = \frac{Gain(A)}{SplitInfo(A)}$$

**Gini index** measures the impurity of a data partition or a set of training tuples. Gini index considers the binary split of each attribute. To determine the best binary split examine all subsets of A excluding the power and empty sets.

Steps for selection using Gini index:

First calculate the impurity of D:

$$Gini(D) = 1 - \sum_{i=1}^{m} p_i^2$$

For each attribute A
{
    for all possible binary splits of A, A partions D into $D_1$ and $D_2$
    {
        Calculate the gini index of that split:

$$Gini_A(D) = \frac{|D_1|}{|D|} Gini(D_1) + \frac{|D_2|}{|D|} Gini(D_2)$$

    }
    Choose the binary split with the lowest gini index.
    Calculate the change in impurity for the lowest gini index:
        (The symbol for “change in” is delta(triangle))

$$\Delta Gini(A) = Gini(D) - Gini_A(D)$$

}
Select the attribute that maximizes the reduction in impurity and split on.

# Clustering

## k-means

Step 1: Select a value for K
- This step is to determine how many clusters we want to form.

Step 2:Select K points to act as out initial centroids
- we select K points so that we can begin to assign our points to their cluster.

Step 3:Assign each of the points to the cluster of its nearest centroids
- The goal of this step is to create our k clusters

Step 4:recalculate the centroids
- as our centroids are no longer the true centroid value we need to recalculate the centroids
- 

Step 5:repeat steps 3 and 4 until the centroids no longer move

The process terminates when we find the values of the centroids from step 3 have not moved once we recalculated them in step 4 i.e. Their value was the same, no points moved cluster.

## AHC

Step 1: Assign each object to its ow single-object cluster Calculate the distance between each pair of clusters.

Step 2: Choose the closest pair of clusters and merge them into a single cluster

Step 3: Calculate the distance between the new cluster and each of the old clusters

Step 4: Repeat steps 2 and 3 until all the objects are in a single cluster.

Single-link Clustering: The distance between two clusters is taken as the **shortest** distance from any member of one to any member of the other.

Complete-link clustering: The distance is taken as the **longest**

Average-link clustering : The distance is taken as the **average** distance.

**Multi-dimensional modelling**

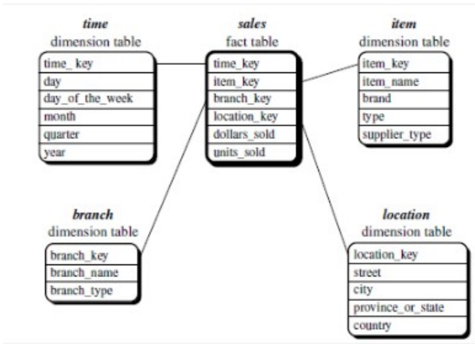
## Fact table

- Located in the center of star and snowflake schemas.
- When multiple are used they are arranges in a fact constellation.
- Typically two types of columns, dimension foreign keys and facts/measures.

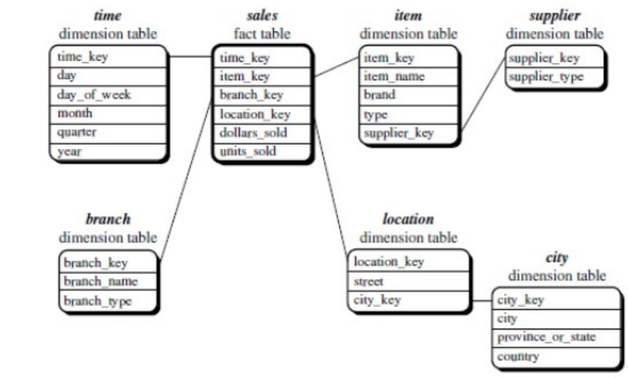
## Dimensions table

- Stores attributes to describe objects in fact table.
- Categorise and describe data warehouse facts and measures in a way to support meaningful answers to business questions.
- Relates to fact table on Primary → Foreign Key relationship

## Star Schema

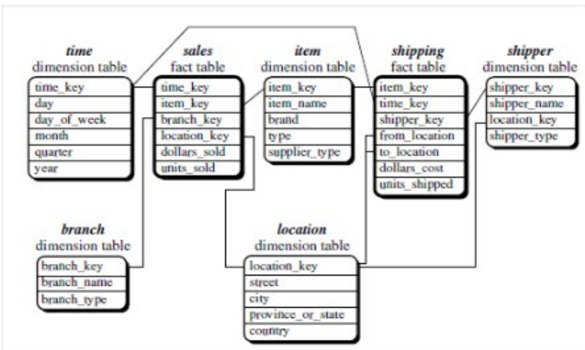



## Snowflake schema




- Variant of Star where some dimension tables are normalized further spliting the data into additional tables.

## Fact Constellation Schema




- Multiple fact tables.

**Concept Hierarchy**

- Defines a sequence of mappings from a set of low-level concepts to higher-level, more general concepts.

**OLAP Operations**

- **Roll up:** Summarize data by climbing up hierarchy or by dimension reduction.
- **Drill Down:** Reverse of roll up from higher-level sumarry to lower level or detailed data, or introducin new dimensions.
- **Slice and dice :** Project and select
- **Pivot:** Reorient the cube, visulization, SD to series of 2D planes
- **Drill across: i**nvolving(across) more then one fact table.
- **Drill Through:** Through the bottom level of the cube to its back-end relational tables(using SQL)