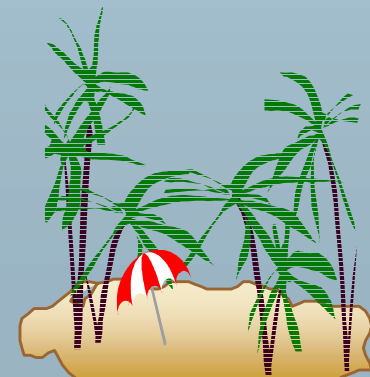
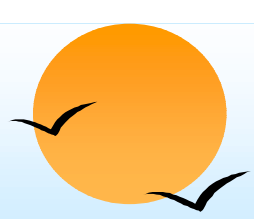


Chapter 22: Advanced Querying and Information Retrieval

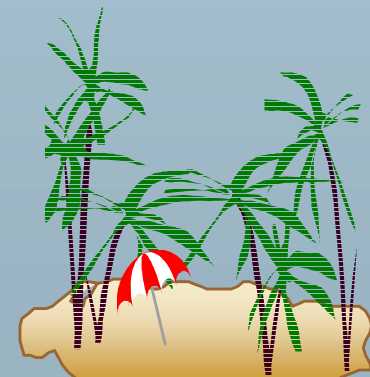
- Decision-Support Systems
 - ☞ Data Analysis
 - 📄 OLAP
 - 📄 Extended aggregation features in SQL
 - Windowing and ranking
 - ☞ Data Mining
 - ☞ Data Warehousing
- Information-Retrieval Systems
 - ☞ Including Web search





Decision Support Systems

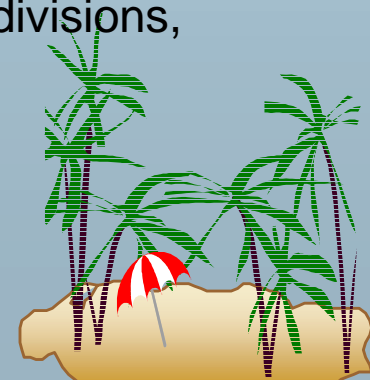
- **Decision-support systems** are used to make business decisions often based on data collected by on-line transaction-processing systems.
- Examples of business decisions:
 - ☞ what items to stock?
 - ☞ What insurance premium to change?
 - ☞ Who to send advertisements to?
- Examples of data used for making decisions
 - ☞ Retail sales transaction details
 - ☞ Customer profiles (income, age, sex, etc.)

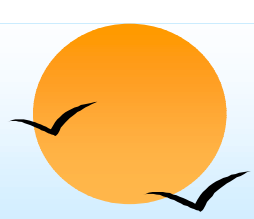




Decision-Support Systems: Overview

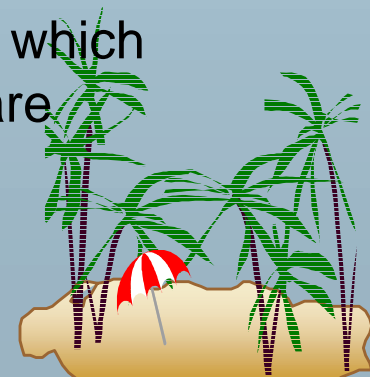
- **Data analysis** tasks are simplified by specialized tools and SQL extensions
 - ☞ Example tasks
 - 📄 For each product category and each region, what were the total sales in the last quarter and how do they compare with the same quarter last year
 - 📄 As above, for each product category and each customer category
- **Statistical analysis** packages (e.g., : S++) can be interfaced with databases
 - ☞ Statistical analysis is a large field will not study it here
- **Data mining** seeks to discover knowledge automatically in the form of statistical rules and patterns from Large databases.
- A **data warehouse** archives information gathered from multiple sources, and stores it under a unified schema, at a single site.
 - ☞ Important for large businesses which generate data from multiple divisions, possibly at multiple sites
 - ☞ Data may also be purchased externally





Data Analysis and OLAP

- Aggregate functions summarize large volumes of data
- **Online Analytical Processing (OLAP)**
 - ☞ Interactive analysis of data, allowing data to be summarized and viewed in different ways in an online fashion (with negligible delay)
- Data that can be modeled as dimension attributes and measure attributes are called **multidimensional data**.
 - ☞ Given a relation used for data analysis, we can identify some of its attributes as **measure attributes**, since they measure some value, and can be aggregated upon. For instance, the attribute *number* of the *sales* relation is a measure attribute, since it measures the number of units sold.
 - ☞ Some of the other attributes of the relation are identified as **dimension attributes**, since they define the dimensions on which measure attributes, and summaries of measure attributes, are viewed.



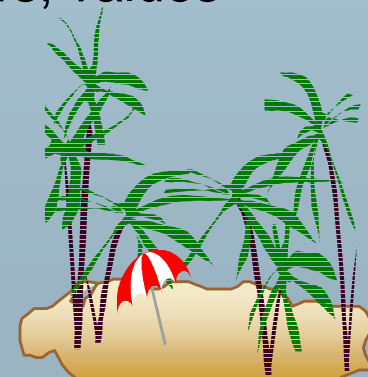
Cross Tabulation of sales by *item-name* and *color*

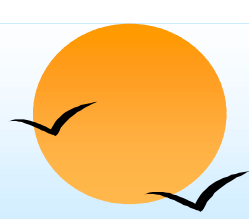
size:

color

	dark	pastel	white	Total
<i>item-name</i>				
skirt	8	35	10	53
dress	20	10	5	35
shirt	14	7	28	49
pant	20	2	5	27
Total	62	54	48	164

- The table above is an example of a **cross-tabulation** (**cross-tab**), also referred to as a **pivot-table**.
- A cross-tab is a table where
 - 👉 values for one of the dimension attributes form the row headers, values for another dimension attribute form the column headers
 - 📄 Other dimension attributes are listed on top
 - 👉 Values in individual cells are (aggregates of) the values of the dimension attributes that specify the cell.

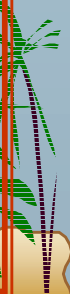


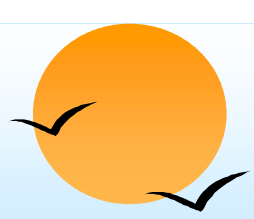


Relational Representation of Crosstabs

- Crosstabs can be represented as relations
 - The value **all** is used to represent aggregates
 - The SQL:1999 standard actually uses null values in place of **all**
 - More on this later....

<i>item-name</i>	<i>color</i>	<i>number</i>
skirt	dark	8
skirt	pastel	35
skirt	white	10
skirt	all	53
dress	dark	20
dress	pastel	10
dress	white	5
dress	all	35
shirt	dark	14
shirt	pastel	7
shirt	white	28
shirt	all	49
pant	dark	20
pant	pastel	2
pant	white	5
pant	all	27
all	dark	62
all	pastel	54
all	white	48
all	all	164

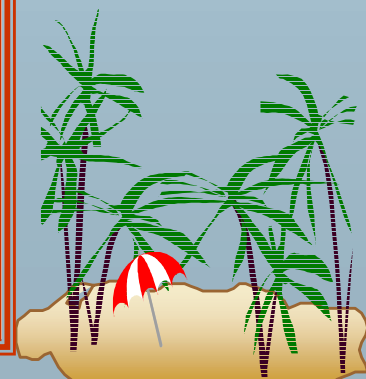


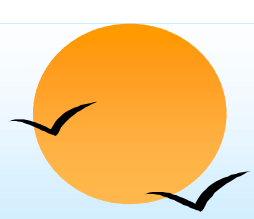


Three-Dimensional Data Cube

- A **data cube** is a multidimensional generalization of a crosstab
 - Cannot view a three-dimensional object in its entirety
 - but crosstabs can be used as views on a data cube

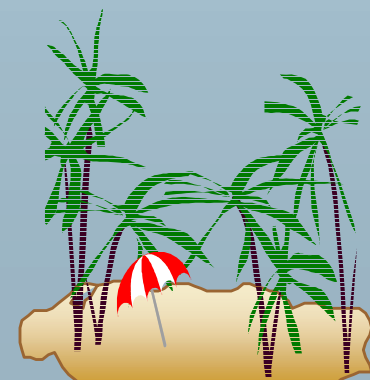
color	item name					size		
	skirt	dress	shirts	pant	all	small	medium	large
dark	8	20	14	20	62	4	18	16
pastel	35	10	7	2	54	9	21	34
white	10	8	28	5	48	42	77	22
all	53	35	49	27	164	55	118	72

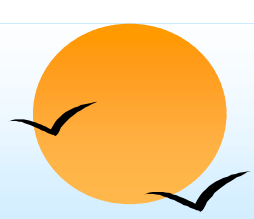




Online Analytical Processing

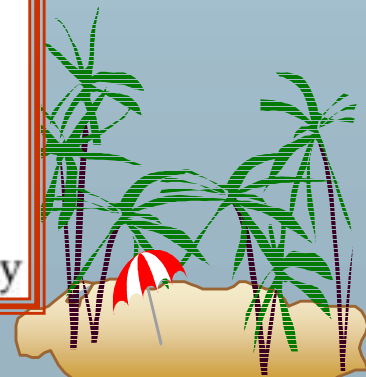
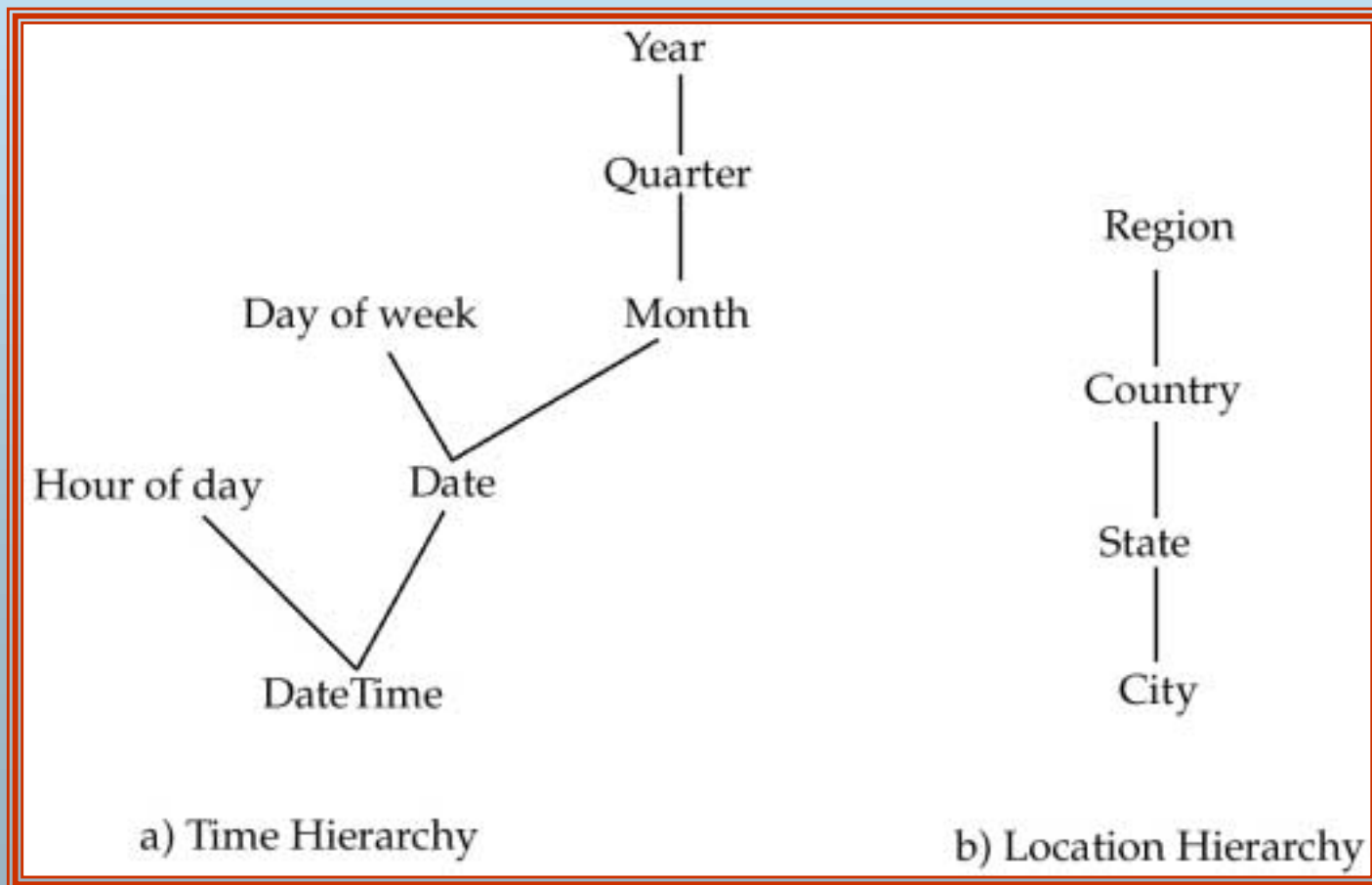
- The operation of changing the dimensions used in a cross-tab is called **pivoting**
- Suppose an analyst wishes to see a cross-tab on *item-name* and *color* for a fixed value of *size*, for example, large, instead of the sum across all sizes.
 - ☞ Such an operation is referred to as **slicing**.
 - 📄 The operation is sometimes called **dicing**, particularly when values for multiple dimensions are fixed.
- The operation of moving from finer-granularity data to a coarser granularity is called a **rollup**.
- The opposite operation - that of moving from coarser-granularity data to finer-granularity data – is called a **drill down**.

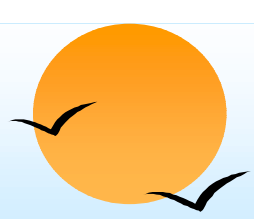




Hierarchies on Dimensions

- **Hierarchy** on dimension attributes: lets dimensions to be viewed at different levels of detail
 - 👉 E.g. the dimension DateTime can be used to aggregate by hour of day, date, day of week, month, quarter or year

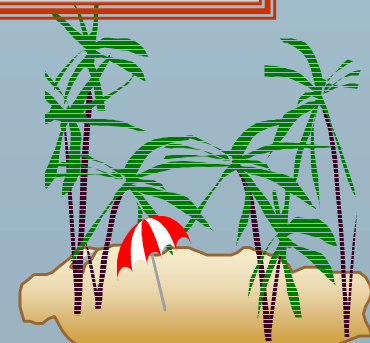


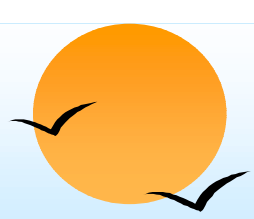


Cross Tabulation With Hierarchy

- Crosstabs can be easily extended to deal with hierarchies
 - ☞ Can drill down or roll up on a hierarchy

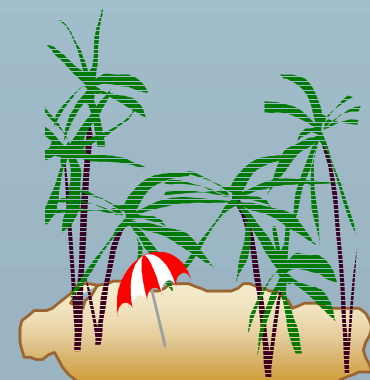
<i>category</i>	<i>item-name</i>	dark	pastel	white	total	
womenswear	skirt	8	8	10	53	
	dress	20	20	5	35	
	subtotal	28	28	15		88
menswear	pants	14	14	28	49	
	shirt	20	20	5	27	
	subtotal	34	34	33		76
total		62	62	48		164

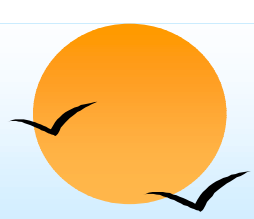




OLAP Implementation

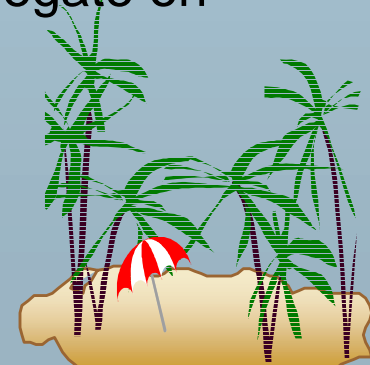
- The earliest OLAP systems used multidimensional arrays in memory to store data cubes, and are referred to as **multidimensional OLAP (MOLAP)** systems.
- OLAP implementations using only relational database features are called **relational OLAP (ROLAP)** systems
- Hybrid systems, which store some summaries in memory and store the base data and other summaries in a relational database, are called **hybrid OLAP (HOLAP)** systems.

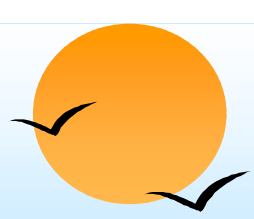




OLAP Implementation (Cont.)

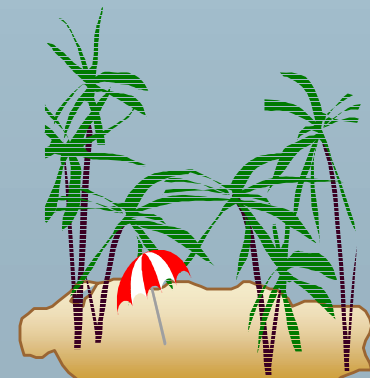
- Early OLAP systems precomputed *all* possible aggregates in order to provide online response
 - ☞ Space and time requirements for doing so can be very high
 - 📄 2^n combinations of **group by**
 - ☞ It suffices to precompute some aggregates, and compute others on demand from one of the precomputed aggregates
 - 📄 Can compute aggregate on (*item-name, color*) from an aggregate on (*item-name, color, size*)
 - For all but a few “non-decomposable” aggregates such as *median*
 - is cheaper than computing it from scratch
- Several optimizations available for computing multiple aggregates
 - ☞ Can compute aggregate on (*item-name, color*) from an aggregate on (*item-name, color, size*)
 - ☞ Can compute aggregates on (*item-name, color, size*), (*item-name, color*) and (*item-name*) using a single sorting of the base data

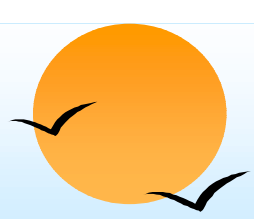




Extended Aggregation

- SQL-92 aggregation quite limited
 - ☞ Many useful aggregates are either very hard or impossible to specify
 - ☞ Data cube
 - ☞ Complex aggregates (median, variance)
 - ☞ binary aggregates (correlation, regression curves)
 - ☞ ranking queries (“assign each student a rank based on the total marks”)
- SQL:1999 OLAP extensions provide a variety of aggregation functions to address above limitations
 - ☞ Supported by several databases, including Oracle and IBM DB2





Extended Aggregation in SQL:1999

- The **cube** operation computes union of **group by**'s on every subset of the specified attributes
- E.g. consider the query

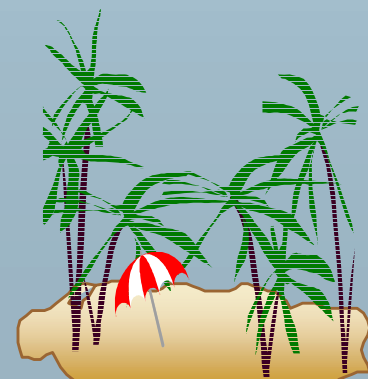
```
select item-name, color, size, sum(number)  
from sales  
group by cube(item-name, color, size)
```

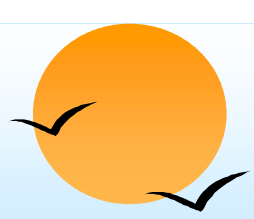
This computes the union of eight different groupings of the *sales* relation:

```
{ (item-name, color, size), (item-name, color),  
  (item-name, size),      (color, size),  
  (item-name),          (color),  
  (size),              ( ) }
```

where () denotes an empty **group by** list.

- For each grouping, the result contains the null value for attributes not present in the grouping.





Extended Aggregation (Cont.)

- Relational representation of crosstab that we saw earlier, but with *null* in place of **all**, can be computed by

```
select item-name, color, sum(number)  
from sales  
group by cube(item-name, color)
```

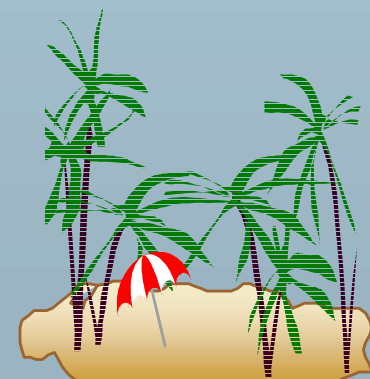
- The function **grouping()** can be applied on an attribute

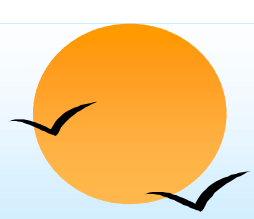
☞ Returns 1 if the value is a null value representing all, and returns 0 in all other cases.

```
select item-name, color, size, sum(number),  
      grouping(item-name) as item-name-flag,  
      grouping(color) as color-flag,  
      grouping(size) as size-flag,  
from sales  
group by cube(item-name, color, size)
```

- Can use the function **decode()** in the **select** clause to replace such nulls by a value such as **all**

☞ E.g. replace *item-name* in first query by
decode(**grouping**(*item-name*), 1, 'all', *item-name*)





Extended Aggregation (Cont.)

- The **rollup** construct generates union on every prefix of specified list of attributes

- E.g.

```
select item-name, color, size, sum(number)  
from sales  
group by rollup(item-name, color, size)
```

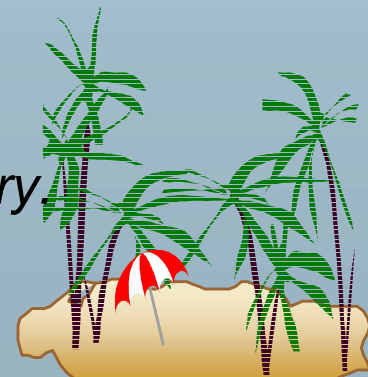
- ☞ Generates union of four groupings:

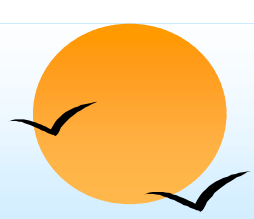
```
{ (item-name, color, size), (item-name, color), (item-name), ( ) }
```

- Rollup can be used to generate aggregates at multiple levels of a hierarchy.
- E.g., suppose table *itemcategory*(*item-name*, *category*) gives the category of each item. Then

```
select category, item-name, sum(number)  
from sales, itemcategory  
where sales.item-name = itemcategory.item-name  
group by rollup(category, item-name)
```

would give a hierarchical summary by *item-name* and by *category*





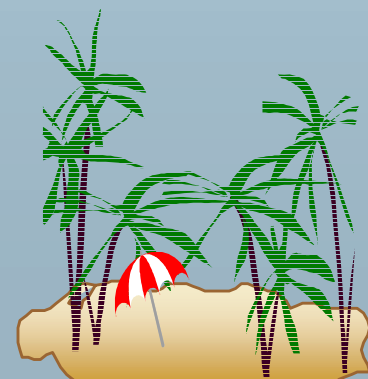
Extended Aggregation (Cont.)

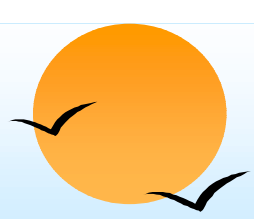
- Multiple rollups and cubes can be used in a single group by clause
 - ☞ Each generates set of group by lists, cross product of sets gives overall set of group by lists

- E.g.,

```
select item-name, color, size, sum(number)  
from sales  
group by rollup(item-name), rollup(color, size)
```

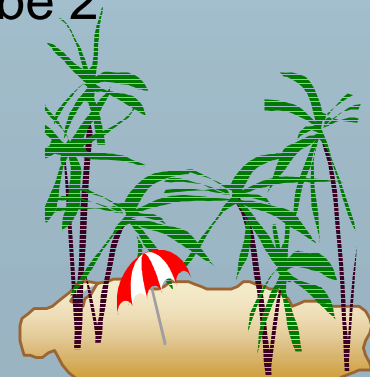
generates the groupings

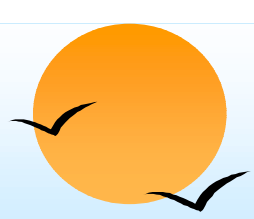
$$\{ \textit{item-name}, () \} \times \{ (\textit{color}, \textit{size}), (\textit{color}), () \}$$
$$= \{ (\textit{item-name}, \textit{color}, \textit{size}), (\textit{item-name}, \textit{color}), (\textit{item-name}), \\ (\textit{color}, \textit{size}), (\textit{color}), () \}$$




Ranking

- Ranking is done in conjunction with an order by specification.
- Given a relation student-marks(student-id, marks) find the rank of each student.
**select student-id, rank() over (order by marks desc) as s-rank
from student-marks**
- An extra **order by** clause is needed to get them in sorted order
**select student-id, rank () over (order by marks desc) as s-rank
from student-marks
order by s-rank**
- Ranking may leave gaps: e.g. if 2 students have the same top mark, both have rank 1, and the next rank is 3
👉 **dense_rank** does not leave gaps, so next dense rank would be 2



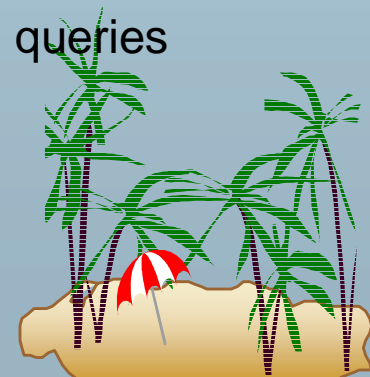


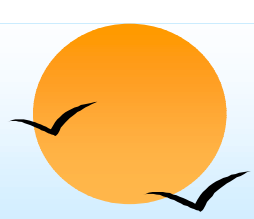
Ranking (Cont.)

- Ranking can be done within partition of the data.
- “Find the rank of students within each section.”

```
select student-id, section,  
        rank ( ) over (partition by section order by marks desc)  
        as sec-rank  
from student-marks, student-section  
where student-marks.student-id = student-section.student-id  
order by section, sec-rank
```

- Multiple **rank** clauses can occur in a single **select** clause
- Ranking is done *after* applying **group by** clause/aggregation
- Exercises:
 - ☞ Find students with top n ranks
 - 📄 Many systems provide special (non-standard) syntax for “top-n” queries
 - ☞ Rank students by sum of their marks in different courses
 - 📄 given relation *student-course-marks(student-id, course, marks)*





Ranking (Cont.)

- Other ranking functions:

- 👉 **percent_rank** (within partition, if partitioning is done)

- 👉 **cume_dist** (cumulative distribution)

- 📄 fraction of tuples with preceding values

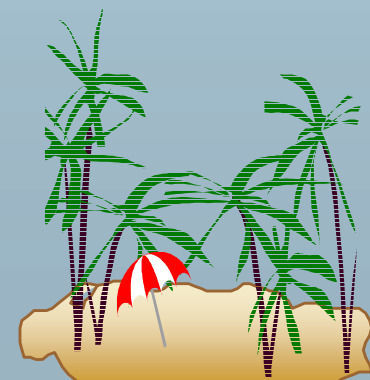
- 👉 **row_number** (non-deterministic in presence of duplicates)

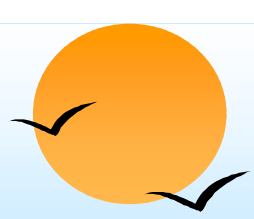
- SQL:1999 permits the user to specify **nulls first** or **nulls last**

- select** *student-id*,

- rank () over (order by *marks desc nulls last*) as *s-rank***

- from** *student-marks*

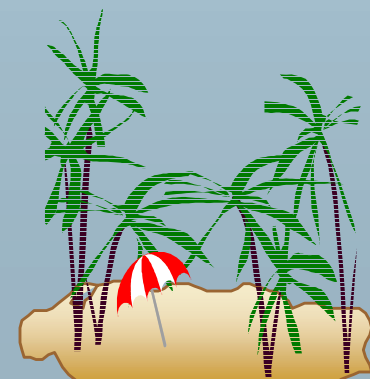


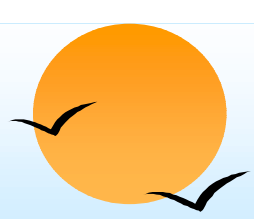


Ranking (Cont.)

- For a given constant n , the ranking the function $ntile(n)$ takes the tuples in each partition in the specified order, and divides them into n buckets with qual numbers of tuples. For instance, we can sort employees by salary, and use $ntile(3)$ to find which range (bottom third, middle third, or top third) each employee is in, and compute the total salary earned by employees in each range:

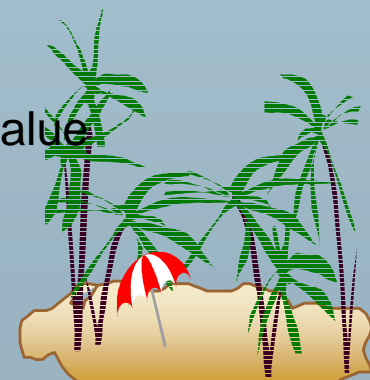
```
select threetile, sum(salary)  
from (  
    select salary, ntile(3) over (order by salary) as threetile  
    from employee) as s  
group by threetile
```

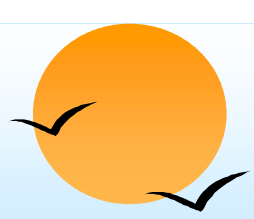




Windowing

- E.g.: “Given sales values for each date, calculate for each date the average of the sales on that day, the previous day, and the next day”
- Such *moving average* queries are used to smooth out random variations.
- In contrast to group by, the same tuple can exist in multiple windows
- **Window specification** in SQL:
 - ☞ Ordering of tuples, size of window for each tuple, aggregate function
 - ☞ E.g. given relation *sales(date, value)*
select date, *sum*(value) over
(order by date between rows 1 preceding and 1 following)
from sales
- Examples of other window specifications:
 - ☞ **between rows unbounded preceding and current**
 - ☞ **rows unbounded preceding**
 - ☞ **range between 10 preceding and current row**
 - 📄 All rows with values between current row value -10 to current value
 - ☞ **range interval 10 day preceding**
 - 📄 Not including current row

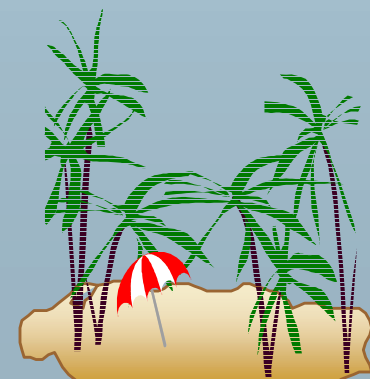




Windowing (Cont.)

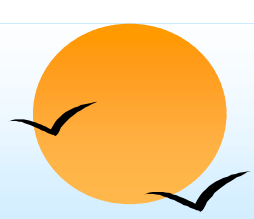
- Can do windowing within partitions
- E.g. Given a relation *transaction*(*account-number*, *date-time*, *value*), where *value* is positive for a deposit and negative for a withdrawal
 - ☞ “Find total balance of each account after each transaction on the account”

```
select account-number, date-time,  
       sum(value) over  
       (partition by account-number  
        order by date-time  
        rows unbounded preceding)  
       as balance  
from transaction  
order by account-number, date-time
```



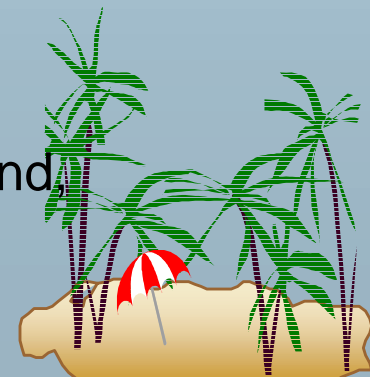
Data Mining

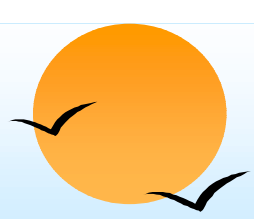




Data Mining

- Broadly speaking, data mining is the process of semi-automatically analyzing large databases to find useful patterns
- Like knowledge discovery in artificial intelligence data mining discovers statistical rules and patterns
- Differs from machine learning in that it deals with large volumes of data stored primarily on disk.
- Some types of knowledge discovered from a database can be represented by a set of rules.
 - ☞ e.g.,: “Young women with annual incomes greater than \$50,000 are most likely to buy sports cars”
- Other types of knowledge represented by equations, or by prediction functions
- Some manual intervention is usually required
 - ☞ Pre-processing of data, choice of which type of pattern to find, postprocessing to find novel patterns





Applications of Data Mining

■ **Prediction** based on past history

- ☞ Predict if a credit card applicant poses a good credit risk, based on some attributes (income, job type, age, ..) and past history
- ☞ Predict if a customer is likely to switch brand loyalty
- ☞ Predict if a customer is likely to respond to “junk mail”
- ☞ Predict if a pattern of phone calling card usage is likely to be fraudulent

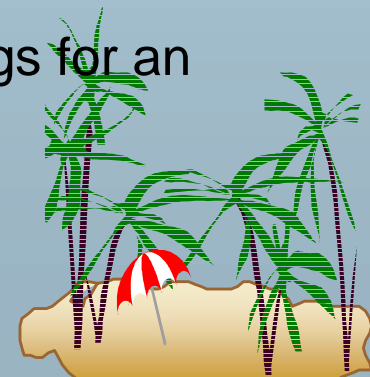
■ Some examples of prediction mechanisms:

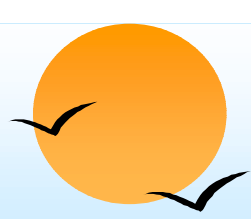
☞ **Classification**

- ☞ Given a training set consisting of items belonging to different classes, and a new item whose class is unknown, predict which class it belongs to

☞ **Regression** formulae

- ☞ given a set of parameter-value to function-result mappings for an unknown function, predict the function-result for a new parameter-value






Applications of Data Mining (Cont.)


■ Descriptive Patterns

Associations


 Find books that are often bought by the same customers. If a new customer buys one such book, suggest that he buys the others too.

 Other similar applications: camera accessories, clothes, etc.

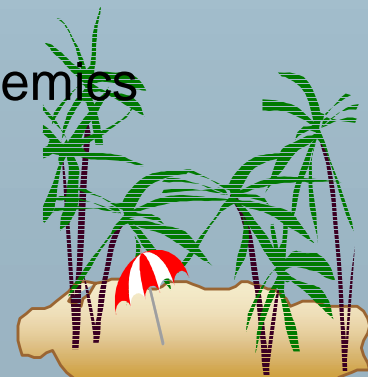
 Associations may also be used as a first step in detecting **causation**

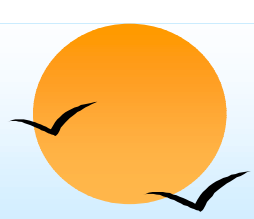
 E.g. association between exposure to chemical X and cancer, or new medicine and cardiac problems

Clusters

 E.g. typhoid cases were clustered in an area surrounding a contaminated well

 Detection of clusters remains important in detecting epidemics

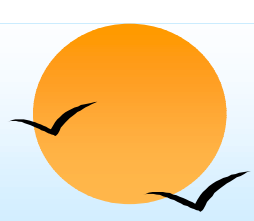




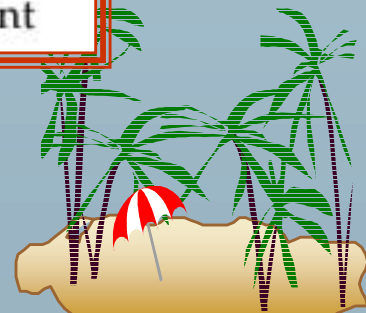
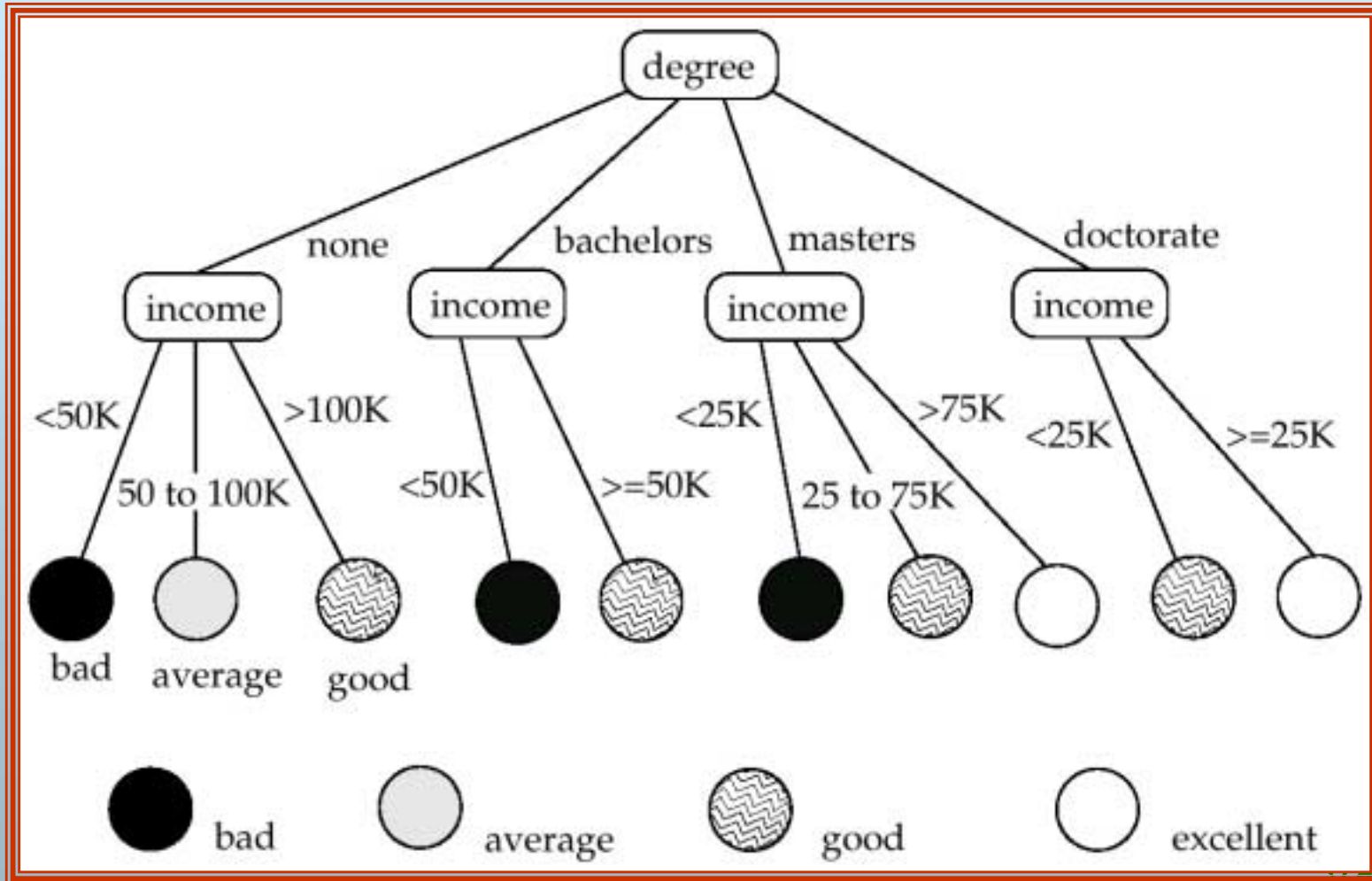
Classification Rules

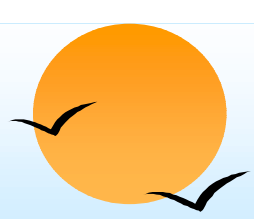
- Classification rules help assign new objects to a set of classes. E.g., given a new automobile insurance applicant, should he or she be classified as low risk, medium risk or high risk?
- Classification rules for above example could use a variety of knowledge, such as educational level of applicant, salary of applicant, age of applicant, etc.
 - ☞ \forall person P, P.degree = masters **and** P.income > 75,000
 \Rightarrow P.credit = excellent
 - ☞ \forall person P, P.degree = bachelors **and**
(P.income \geq 25,000 and P.income \leq 75,000)
 \Rightarrow P.credit = good
- Rules are not necessarily exact: there may be some misclassifications
- Classification rules can be compactly shown as a decision tree.





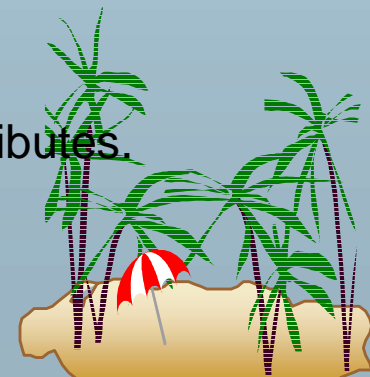
Decision Tree

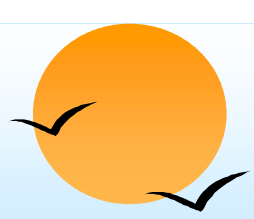




Construction of Decision Trees

- **Training set:** a data sample in which the grouping for each tuple is already known.
- Consider credit risk example: Suppose *degree* is chosen to partition the data at the root.
 - ☞ Since *degree* has a small number of possible values, one child is created for each value.
- At each child node of the root, further classification is done if required. Here, partitions are defined by *income*.
 - ☞ Since *income* is a continuous attribute, some number of intervals are chosen, and one child created for each interval.
- Different classification algorithms use different ways of choosing which attribute to partition on at each node, and what the intervals, if any, are.
- In general
 - ☞ Different branches of the tree could grow to different levels.
 - ☞ Different nodes at the same level may use different partitioning attributes.



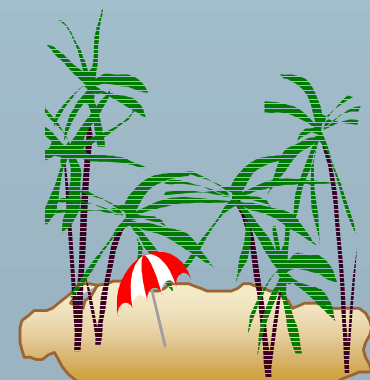


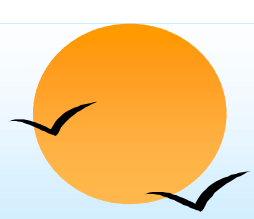
Construction of Decision Trees (Cont.)

■ Greedy top down generation of decision trees.

- ☞ Each internal node of the tree partitions the data into groups based on a **partitioning attribute**, and a **partitioning condition** for the node
 - 📄 More on choosing partitioning attribute/condition shortly
 - 📄 Algorithm is greedy: the choice is made once and not revisited as more of the tree is constructed
- ☞ The data at a node is not partitioned further if either
 - 📄 all (or most) of the items at the node belong to the same class, or
 - 📄 all attributes have been considered, and no further partitioning is possible.

Such a node is a leaf node.
- ☞ Otherwise the data at the node is partitioned further by picking an attribute for partitioning data at the node.



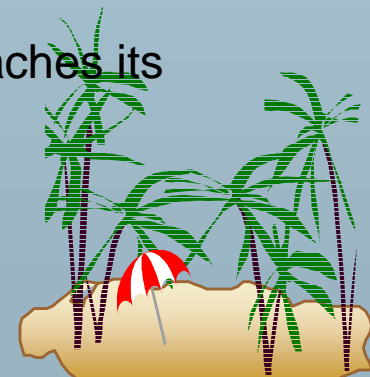


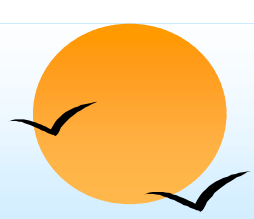
Best Splits

- Idea: evaluate different attributes and partitioning conditions and pick the one that best improves the “purity” of the training set examples
 - 👉 The initial training set has a mixture of instances from different classes and is thus relatively impure
 - 👉 E.g. if degree exactly predicts credit risk, partitioning on degree would result in each child having instances of only one class
 - 📄 I.e., the child nodes would be *pure*
- The purity of a set S of training instances can be measured quantitatively in several ways.
- Notation: number of classes = k , number of instances = $|S|$, fraction of instances in class $i = p_i$
- The **Gini** measure of purity is defined as

$$\text{Gini}(S) = 1 - \sum_{i=1}^k p_i^2$$

- 👉 When all instances are in a single class, the Gini value is 0, while it reaches its maximum (of $1 - 1/k$) if each class the same number of instances.





Best Splits (Cont.)

- Another measure of purity is the **entropy** measure, which is defined as

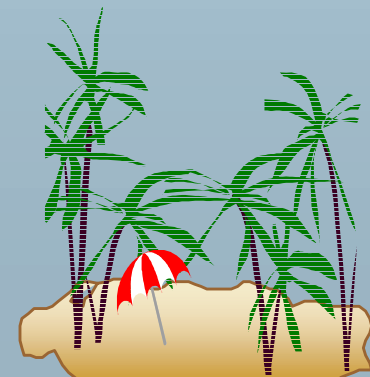
$$\text{entropy}(S) = - \sum_{i=1}^k p_i \log_2 p_i$$

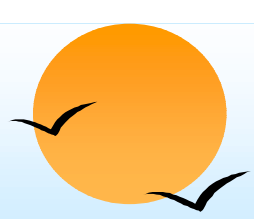
- When a set S is split into multiple sets S_i , $i=1, 2, \dots, r$, we can measure the purity of the resultant set of sets as:

$$\text{purity}(S_1, S_2, \dots, S_r) = \sum_{i=1}^r \frac{|S_i|}{|S|} \text{purity}(S_i)$$

- The information gain due to particular split of S into S_i , $i = 1, 2, \dots, r$

$$\text{Information-gain}(S, \{S_1, S_2, \dots, S_r\}) = \text{purity}(S) - \text{purity}(S_1, S_2, \dots, S_r)$$





Best Splits (Cont.)

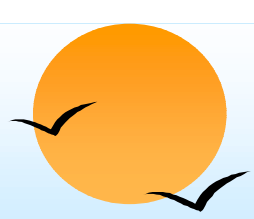
- Measure of “cost” of a split:

Information-content($S, \{S_1, S_2, \dots, S_r\}$) =

$$-\sum_{i=1}^r \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$$

- **Information-gain ratio** =
$$\frac{\text{Information-gain}(S, \{S_1, S_2, \dots, S_r\})}{\text{Information-content}(S, \{S_1, S_2, \dots, S_r\})}$$
- The best split for an attribute is the one that gives the maximum information gain ratio
- **Continuous valued attributes**
 - ☞ Can be ordered in a fashion meaningful to classification
 - ☞ e.g. integer and real values
- **Categorical attributes**
 - ☞ Cannot be meaningfully ordered (e.g. country, school/university, item-color, .):





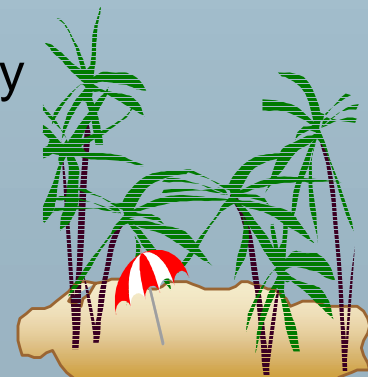
Finding Best Splits

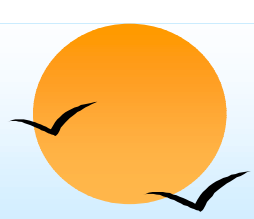
■ Categorical attributes:

- 👉 Multi-way split, one child for each value
 - 📄 may have too many children in some cases
- 👉 Binary split: try all possible breakup of values into two sets, and pick the best

■ Continuous valued attribute

- 👉 Binary split:
 - 📄 Sort values in the instances, try each as a split point
 - E.g. if values are 1, 10, 15, 25, split at ≤ 1 , ≤ 10 , ≤ 15
 - 📄 Pick the value that gives best split
- 👉 Multi-way split: more complicated, see bibliographic notes
 - 📄 A series of binary splits on the same attribute has roughly equivalent effect





Decision-Tree Construction Algorithm

Procedure *Grow.Tree*(S)

 Partition(S);

Procedure Partition (S)

 if ($\text{purity}(S) > \delta_p$ or $|S| < \delta_s$) then

return;

for each attribute A

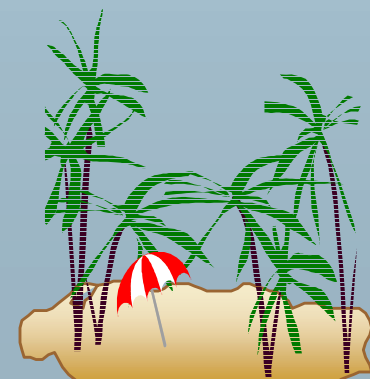
 evaluate splits on attribute A ;

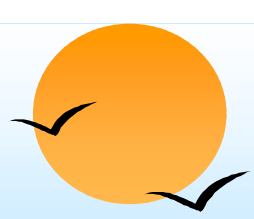
 Use best split found (across all attributes) to partition

S into S_1, S_2, \dots, S_r ,

for $i = 1, 2, \dots, r$

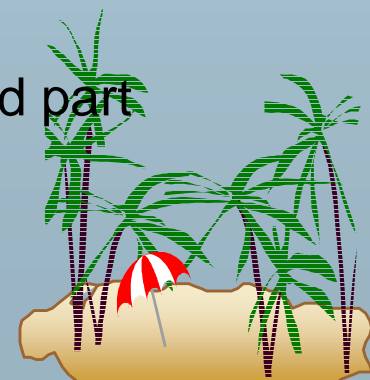
 Partition(S_i);

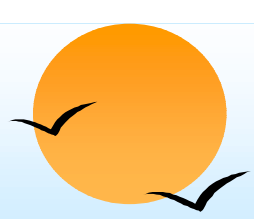




Decision Tree Constr'n Algo's. (Cont.)

- Variety of algorithms have been developed to
 - ☞ Reduce CPU cost and/or
 - ☞ Reduce IO cost when handling datasets larger than memory
 - ☞ Improve accuracy of classification
- Decision tree may be **overfitted**, i.e., overly tuned to given training set
 - ☞ Pruning of decision tree may be done on branches that have too few training instances
 - ☞ When a subtree is pruned, an internal node becomes a leaf
 - and its class is set to the majority class of the instances that map to the node
 - ☞ Pruning can be done by using a part of the training set to build tree, and a second part to test the tree
 - ☞ prune subtrees that increase misclassification on second part





Other Types of Classifiers

- Further types of classifiers
 - ☞ Neural net classifiers
 - ☞ Bayesian classifiers
- Neural net classifiers use the training data to train artificial neural nets
 - ☞ Widely studied in AI, won't cover here
- Bayesian classifiers use **Bayes theorem**, which says

$$p(c_j | d) = \frac{p(d | c_j) p(c_j)}{p(d)}$$

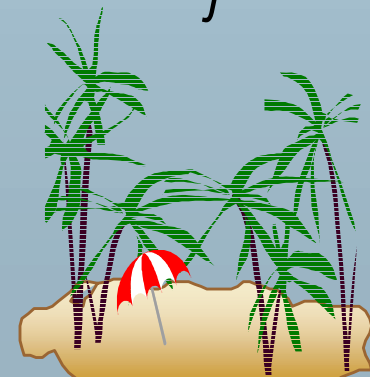
where

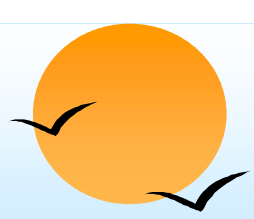
$p(c_j | d)$ = probability of instance d being in class c_j ,

$p(d | c_j)$ = probability of generating instance d given class c_j ,

$p(c_j)$ = probability of occurrence of class c_j , and

$p(d)$ = probability of instance d occurring





Naïve Bayesian Classifiers

- Bayesian classifiers require

- ☞ computation of $p(d | c_j)$

- ☞ precomputation of $p(c_j)$

- ☞ $p(d)$ can be ignored since it is the same for all classes

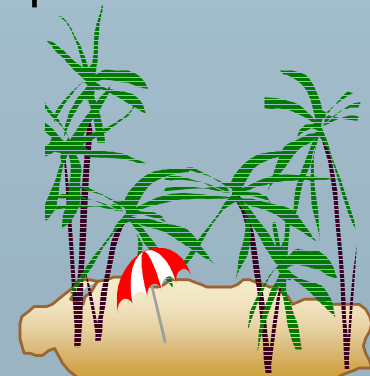
- To simplify the task, **naïve Bayesian classifiers** assume attributes have independent distributions, and thereby estimate

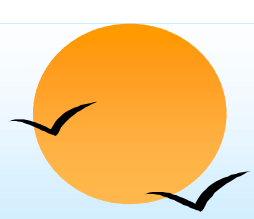
$$p(d|c_j) = p(d_1|c_j) * p(d_2|c_j) * \dots * (p(d_n|c_j))$$

- ☞ Each of the $p(d_i|c_j)$ can be estimated from a histogram on d_i values for each class c_j

- 📄 the histogram is computed from the training instances

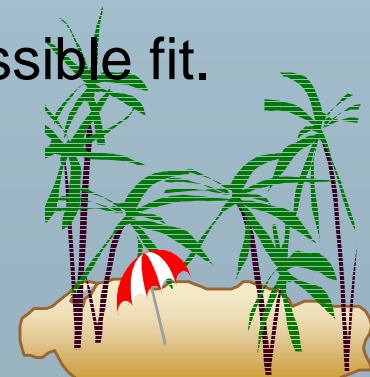
- ☞ Histograms on multiple attributes are more expensive to compute and store

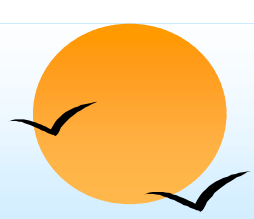




Regression

- Regression deals with the prediction of a value, rather than a class.
 - ☞ Given values for a set of variables, X_1, X_2, \dots, X_n , we wish to predict the value of a variable Y .
- One way is to infer coefficients $a_0, a_1, a_1, \dots, a_n$ such that
$$Y = a_0 + a_1 * X_1 + a_2 * X_2 + \dots + a_n * X_n$$
- Finding such a linear polynomial is called **linear regression**.
 - ☞ In general, the process of finding a curve that fits the data is also called **curve fitting**.
- The fit may only be approximate
 - ☞ because of noise in the data, or
 - ☞ because the relationship is not exactly a polynomial
- Regression aims to find coefficients that give the best possible fit.





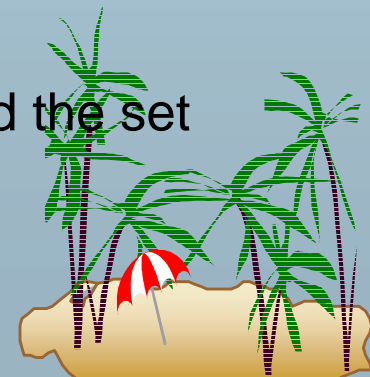
Association Rules

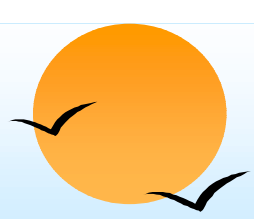
- Retail shops are often interested in associations between different items that people buy.
 - ☞ Someone who buys bread is quite likely also to buy milk
 - ☞ A person who bought the book *Database System Concepts* is quite likely also to buy the book *Operating System Concepts*.
- Associations information can be used in several ways.
 - ☞ E.g. when a customer buys a particular book, an online shop may suggest associated books.

■ Association rules:

bread \Rightarrow *milk* *DB-Concepts, OS-Concepts* \Rightarrow *Networks*

- ☞ Left hand side: **antecedent**, right hand side: **consequent**
- ☞ An association rule must have an associated **population**; the population consists of a set of **instances**
 - ☞ E.g. each transaction (sale) at a shop is an instance, and the set of all transactions is the population

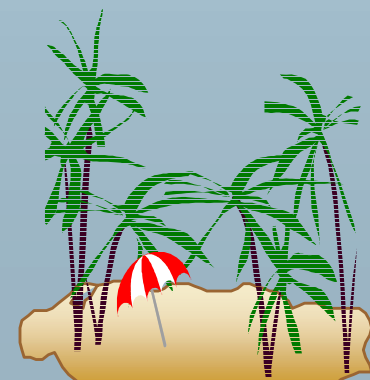


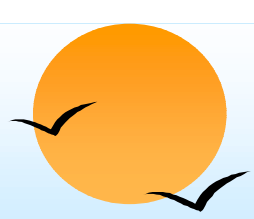


Association Rules (Cont.)

- Rules have an associated support, as well as an associated confidence.
- **Support** is a measure of what fraction of the population satisfies both the antecedent and the consequent of the rule.
 - 👉 E.g. suppose only 0.001 percent of all purchases include milk and screwdrivers. The support for the rule is $milk \Rightarrow screwdrivers$ is low.
 - 👉 We usually want rules with a reasonably high support
 - 📖 Rules with low support are usually not very useful
- **Confidence** is a measure of how often the consequent is true when the antecedent is true.
 - 👉 E.g. the rule $bread \Rightarrow milk$ has a confidence of 80 percent if 80 percent of the purchases that include bread also include milk.
 - 👉 Usually want rules with reasonably large confidence.
 - 📖 A rule with a low confidence is not meaningful.

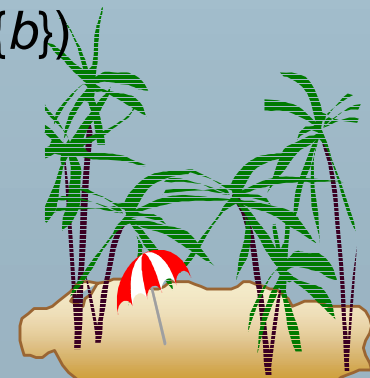
Note that the confidence of $bread \Rightarrow milk$ may be very different from the confidence of $milk \Rightarrow bread$, although both have the same supports.

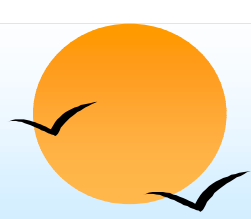




Finding Association Rules

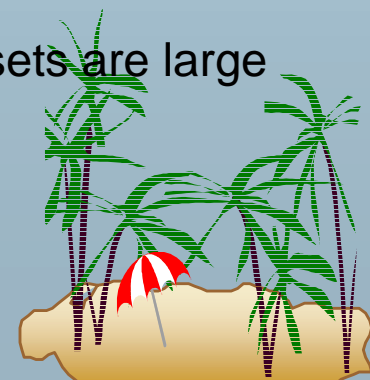
- We are generally only interested in association rules with reasonably high support (e.g. support of 2% or greater)
- Naïve algorithm
 1. Consider all possible sets of relevant items.
 2. For each set find its support (i.e. count how many transactions purchase all items in the set).
 - ✎ **Large itemsets**: sets with sufficiently high support
 3. Use large itemsets to generate association rules.
 1. From itemset A generate the rule $A - \{b\} \Rightarrow b$ for each $b \in A$.
 - 📄 Support of rule = support (A).
 - 📄 Confidence of rule = support (A) / support ($A - \{b\}$)

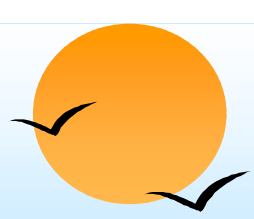




Finding Support

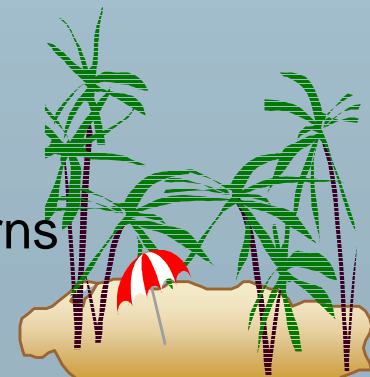
- Few itemsets: determine support of all itemsets via a single pass on set of transactions
 - 👉 A count is maintained for each itemset, initially set to 0.
 - 👉 When a transaction is fetched, the count is incremented for each set of items that is contained in the transaction.
 - 👉 Large itemsets: sets with a high count at the end of the pass
- Many itemsets: If memory not enough to hold all counts for all itemsets use multiple passes, considering only some itemsets in each pass.
- Optimization: Once an itemset is eliminated because its count (support) is too small none of its supersets needs to be considered.
- The **a priori** technique to find large itemsets:
 - 👉 Pass 1: count support of all sets with just 1 item. Eliminate those items with low support
 - 👉 Pass i : **candidates**: every set of i items such that all its $i-1$ item subsets are large
 - 📄 Count support of all candidates
 - 📄 Stop if there are no candidates

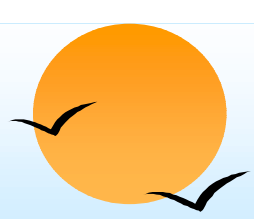




Other Types of Associations

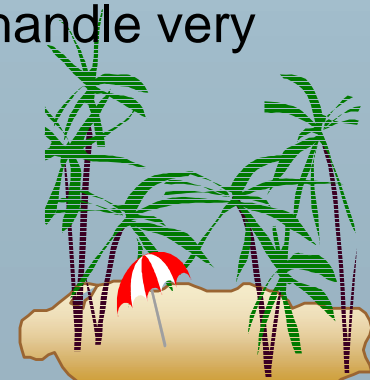
- Basic association rules have several limitations
- Deviations from the expected probability are more interesting
 - ☞ E.g. if many people purchase bread, and many people purchase cereal, quite a few would be expected to purchase both ($\text{prob1} * \text{prob2}$)
 - ☞ We are interested in **positive** as well as **negative correlations** between sets of items
 - ☞ Positive correlation: co-occurrence is higher than predicted
 - ☞ Negative correlation: co-occurrence is lower than predicted
- Sequence associations/correlations
 - ☞ E.g. whenever bonds go up, stock prices go down in 2 days
- Deviations from temporal patterns
 - ☞ E.g. deviation from a steady growth
 - ☞ E.g. sales of winter wear go down in summer
 - ☞ Not surprising, part of a known pattern.
 - ☞ Look for deviation from value predicted using past patterns

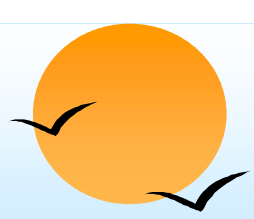




Clustering

- Clustering: Intuitively, finding clusters of points in the given data such that similar points lie in the same cluster
- Can be formalized using distance metrics in several ways
- E.g. Group points into k sets (for a given k) such that the average distance of points from the centroid of their assigned group is minimized
 - ☞ Centroid: point defined by taking average of coordinates in each dimension.
 - ☞ Another metric: minimize average distance between every pair of points in a cluster
- Has been studied extensively in statistics, but on small data sets
 - ☞ Data mining systems aim at clustering techniques that can handle very large data sets
 - ☞ E.g. the Birch clustering algorithm (more shortly)

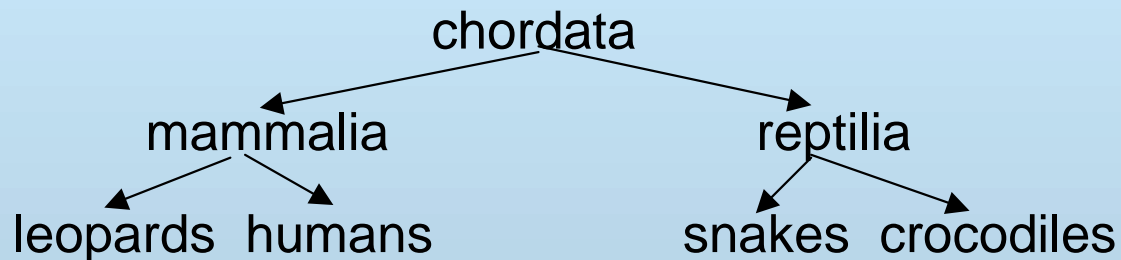




Hierarchical Clustering

- Example from biological classification

- ☞ (the word classification here does not mean a prediction mechanism)



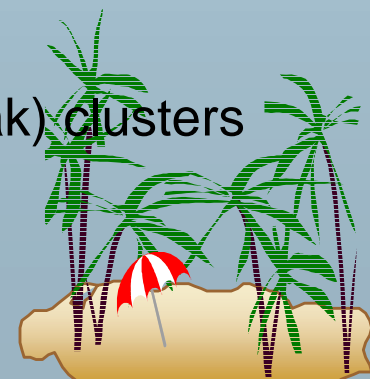
- Other examples: Internet directory systems (e.g. Yahoo, more on this later)

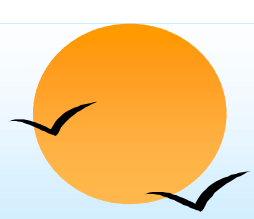
- Agglomerative clustering algorithms

- ☞ Build small clusters, then cluster small clusters into bigger clusters, and so on

- Divisive clustering algorithms

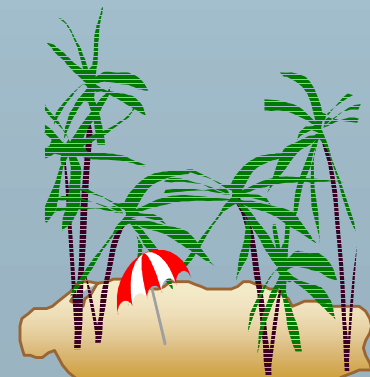
- ☞ Start with all items in a single cluster, repeatedly refine (break) clusters into smaller ones

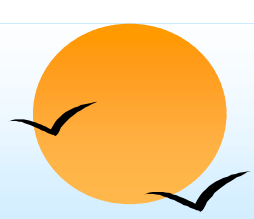




Clustering Algorithms

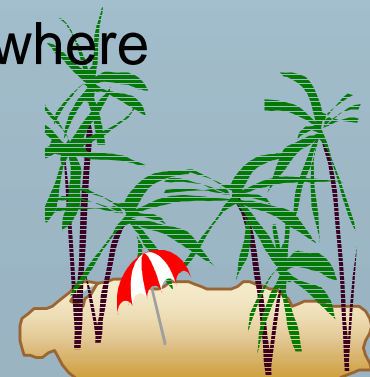
- Clustering algorithms have been designed to handle very large datasets
- E.g. the **Birch algorithm**
 - ☞ Main idea: use an in-memory R-tree to store points that are being clustered
 - ☞ Insert points one at a time into the R-tree, merging a new point with an existing cluster if it is less than some δ distance away
 - ☞ If there are more leaf nodes than fit in memory, merge existing clusters that are close to each other
 - ☞ At the end of first pass we get a large number of clusters at the leaves of the R-tree
 - 📄 Merge clusters to reduce the number of clusters

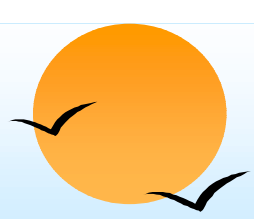




Collaborative Filtering

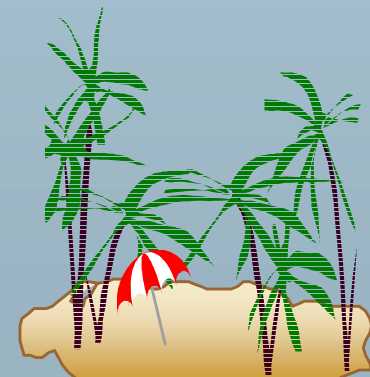
- Goal: predict what movies/books/... a person may be interested in, on the basis of
 - ☞ Past preferences of the person
 - ☞ Other people with similar past preferences
 - ☞ The preferences of such people for a new movie/book/...
- One approach based on repeated clustering
 - ☞ Cluster people on the basis of preferences for movies
 - ☞ Then cluster movies on the basis of being liked by the same clusters of people
 - ☞ Again cluster people based on their preferences for (the newly created clusters of) movies
 - ☞ Repeat above till equilibrium
- Above problem is an instance of **collaborative filtering**, where users collaborate in the task of filtering information to find information of interest





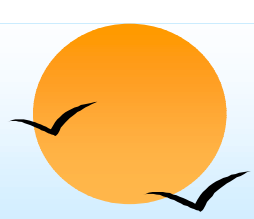
Other Types of Mining

- **Text mining**: application of data mining to textual documents
 - 👉 E.g. cluster Web pages to find related pages
 - 👉 E.g. cluster pages a user has visited to organize their visit history
 - 👉 E.g. classify Web pages automatically into a Web directory
- **Data visualization** systems help users examine large volumes of data and detect patterns visually
 - 👉 E.g. maps, charts, and color-coding
 - 📄 E.g. locations with problems shown in red on a map
 - 👉 Can visually encode large amounts of information on a single screen
 - 👉 Humans are very good at detecting visual patterns



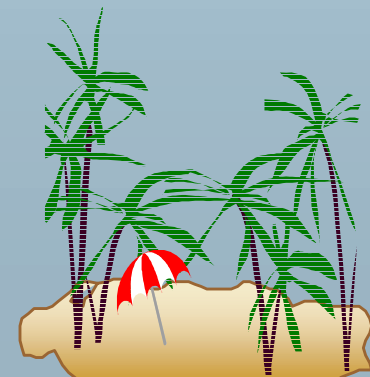
Data Warehousing

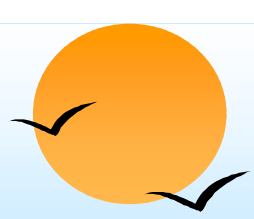




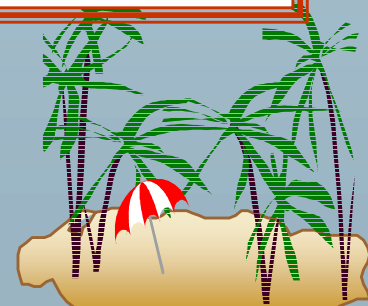
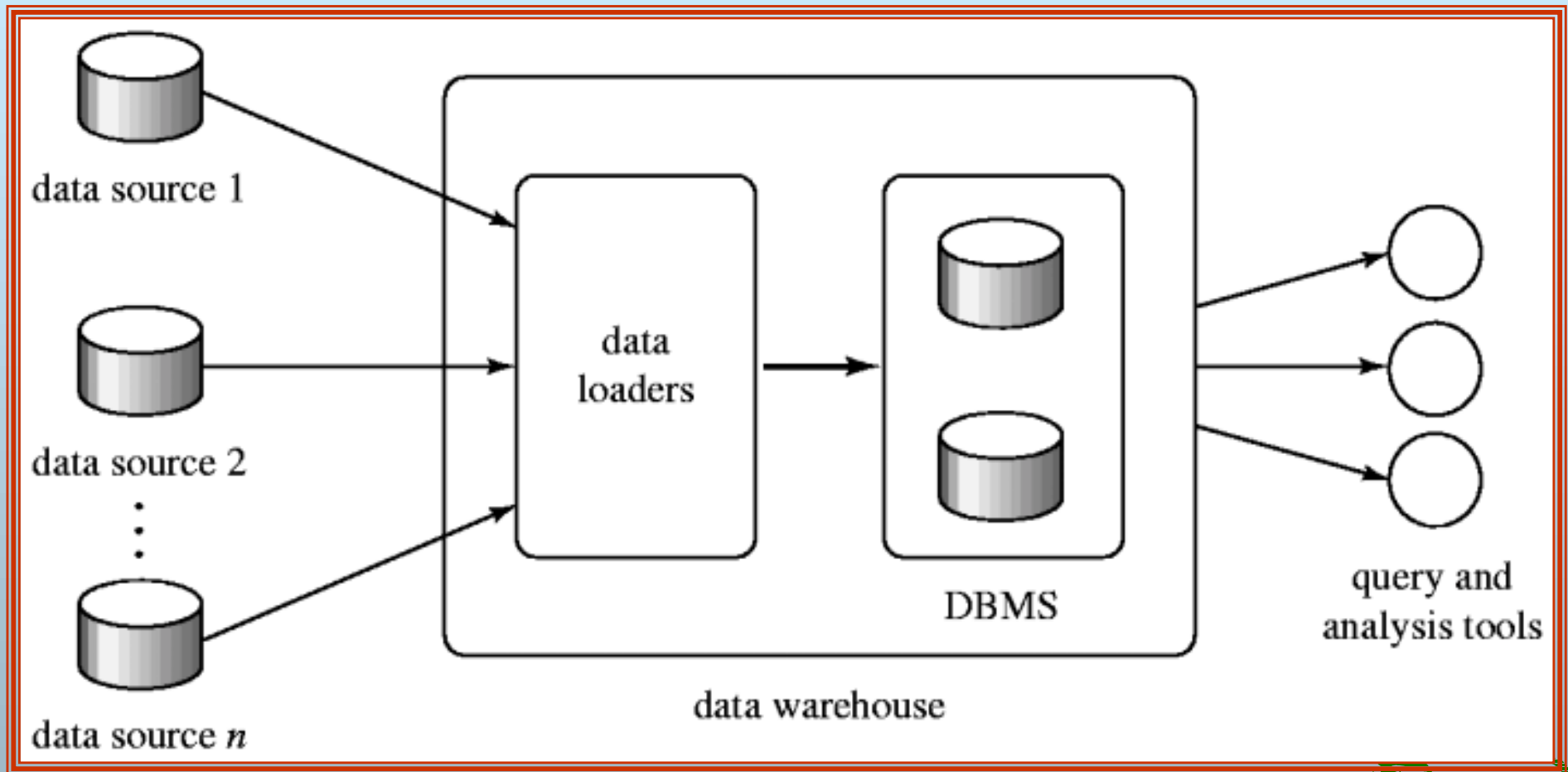
Data Warehousing

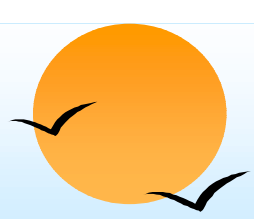
- Large organizations have complex internal organizations, and have data stored at different locations, on different operational (transaction processing) systems, under different schemas
- Data sources often store only current data, not historical data
- Corporate decision making requires a unified view of all organizational data, including historical data
- A **data warehouse** is a repository (archive) of information gathered from multiple sources, stored under a unified schema, at a single site
 - ☞ Greatly simplifies querying, permits study of historical trends
 - ☞ Shifts decision support query load away from transaction processing systems





Data Warehousing





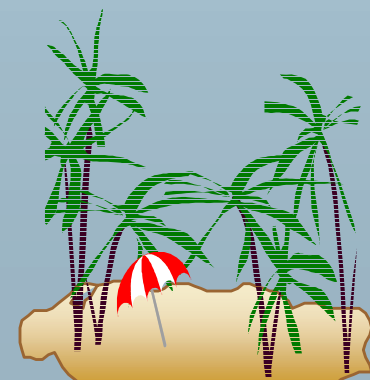
Components of Data Warehouse

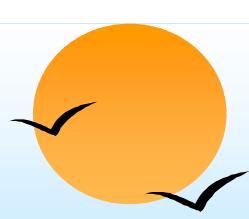
■ *When and how to gather data*

- ☞ **Source driven architecture:** data sources transmit new information to warehouse, either continuously or periodically (e.g. at night)
- ☞ **Destination driven architecture:** warehouse periodically requests new information from data sources
- ☞ Keeping warehouse exactly synchronized with data sources (e.g. using two-phase commit) is too expensive
 - 📄 Usually OK to have slightly out-of-date data at warehouse
 - 📄 Data/updates are periodically downloaded from online transaction processing (OLTP) systems.

■ *What schema to use*

- ☞ Schema integration





Components of Data Warehouse (Cont.)

■ *Data cleansing*

☞ E.g. correct mistakes in addresses

📄 E.g. misspellings, zip code errors

☞ **Merge** address lists from different sources and **purge** duplicates

📄 Keep only one address record per household (“**householding**”)

■ *How to propagate updates*

☞ Warehouse schema may be a (materialized) view of schema from data sources

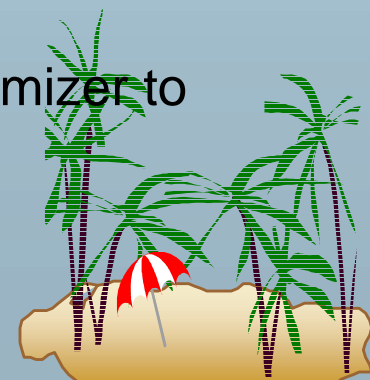
☞ Efficient techniques for update of materialized views

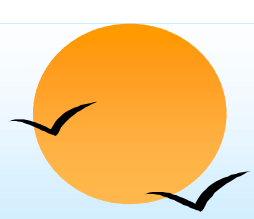
■ *What data to summarize*

☞ Raw data may be too large to store on-line

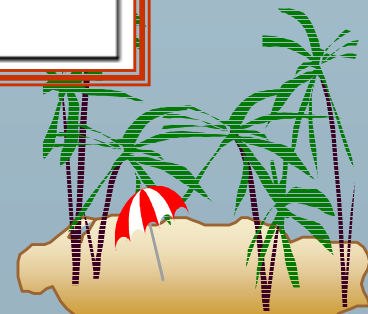
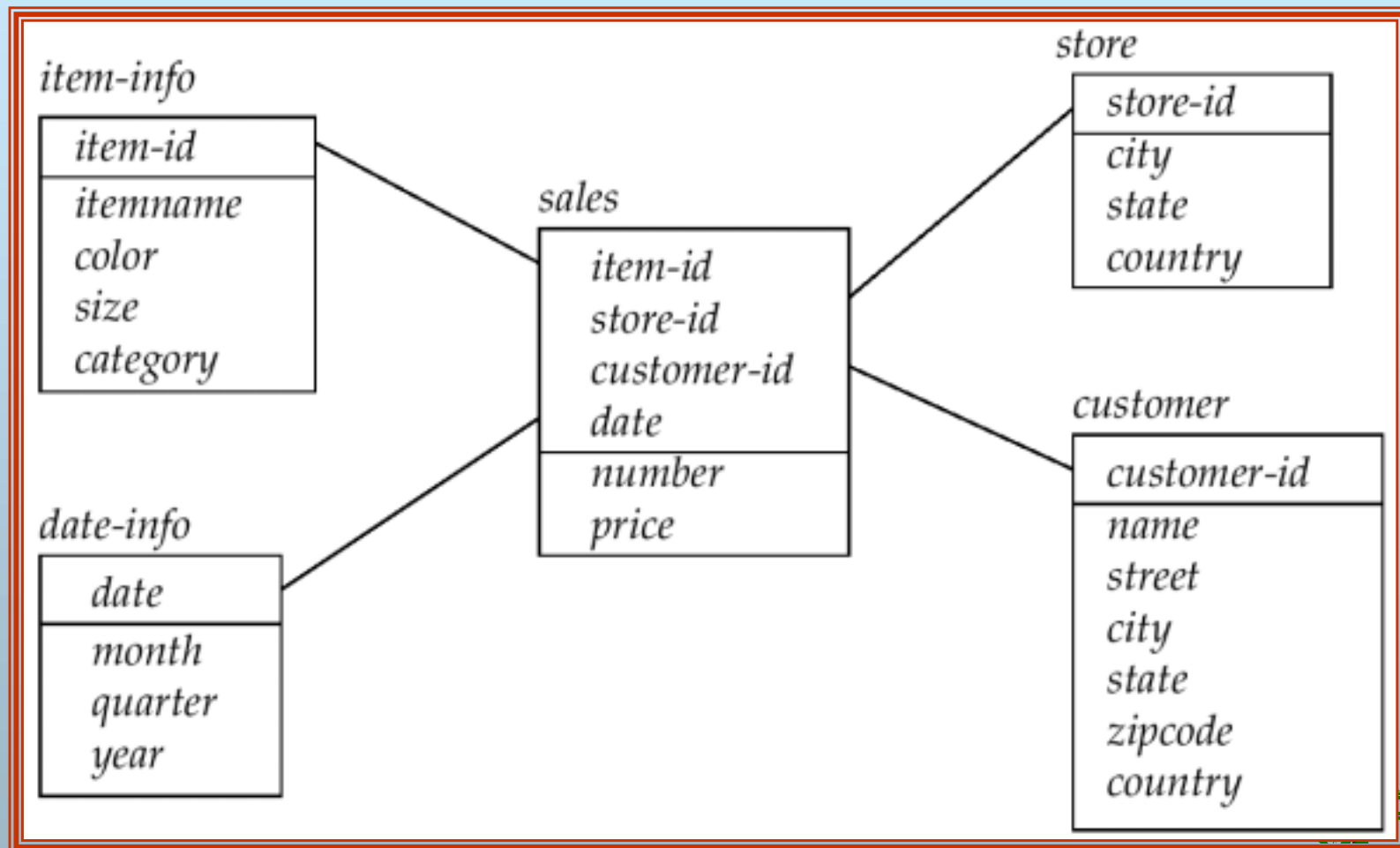
☞ Aggregate values (totals/subtotals) often suffice

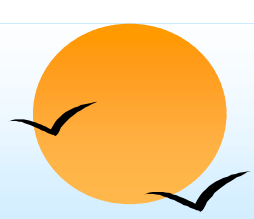
☞ Queries on raw data can often be transformed by query optimizer to use aggregate values





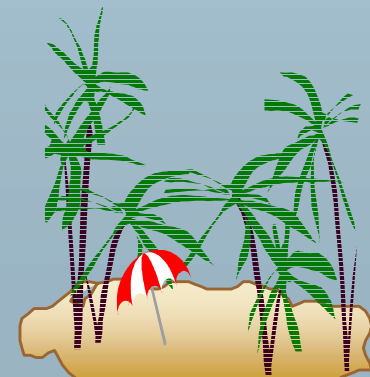
Data Warehouse Schemas





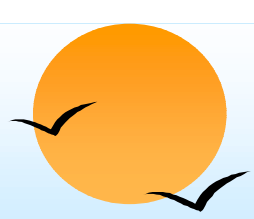
Warehouse Schemas

- Typically warehouse data is multidimensional, with very large **fact tables**
 - ☞ Examples of **dimensions**: item-id, date/time of sale, store where sale was made, customer identifier
 - ☞ Examples of **measures**: number of items sold, price of items
- Dimension values are usually encoded using small integers and mapped to full values via dimension tables
 - ☞ Resultant schema is called a **star schema**
 - 📄 More complicated schema structures
 - **Snowflake schema**: multiple levels of dimension tables
 - **Constellation**: multiple fact tables



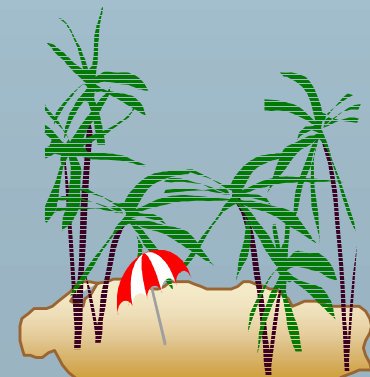
Information Retrieval

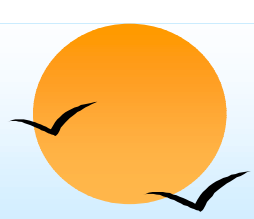




Information Retrieval Systems

- **Information retrieval (IR)** systems use a simpler data model than database systems
 - ☞ Information organized as a collection of documents
 - ☞ Documents are unstructured, no schema
- Information retrieval locates relevant documents, on the basis of user input such as keywords or example documents
 - ☞ e.g., find documents containing the words “database systems”
- Can be used even on textual descriptions provided with non-textual data such as images
- IR on Web documents has become extremely important
 - ☞ E.g. google, altavista, ...

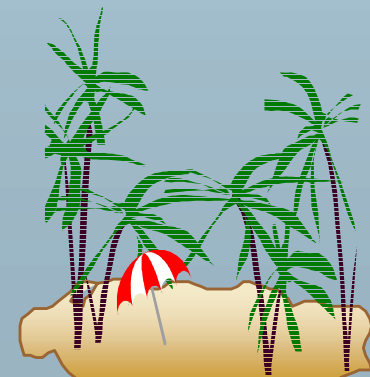


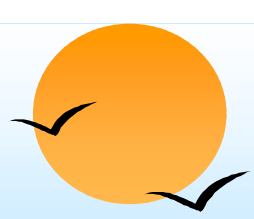


Information Retrieval Systems (Cont.)

■ Differences from database systems

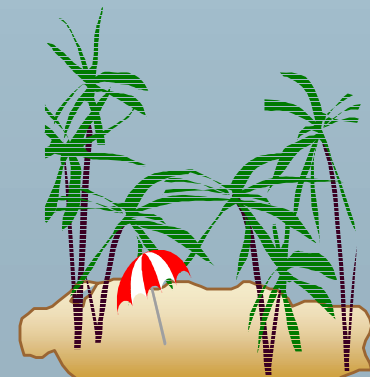
- ☞ IR systems don't deal with transactional updates (including concurrency control and recovery)
- ☞ Database systems deal with structured data, with schemas that define the data organization
- ☞ IR systems deal with some querying issues not generally addressed by database systems
 - 📄 Approximate searching by keywords
 - 📄 Ranking of retrieved answers by estimated degree of relevance

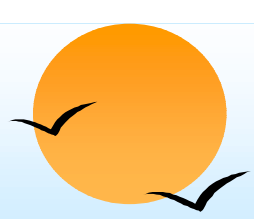




Keyword Search

- In **full text** retrieval, all the words in each document are considered to be keywords.
 - ☞ We use the word **term** to refer to the words in a document
- Information-retrieval systems typically allow query expressions formed using keywords and the logical connectives *and*, *or*, and *not*
 - ☞ *Ands* are implicit, even if not explicitly specified
- Ranking of documents on the basis of estimated relevance to a query is critical
 - ☞ Relevance ranking is based on factors such as
 - 📄 **Term frequency**
 - Frequency of occurrence of query keyword in document
 - 📄 **Inverse document frequency**
 - How many documents the query keyword occurs in
 - » Fewer → give more importance to keyword
 - 📄 **Hyperlinks to documents**
 - More links to a document → document is more important





Relevance Ranking Using Terms

■ TF-IDF (Term frequency/Inverse Document frequency) ranking:

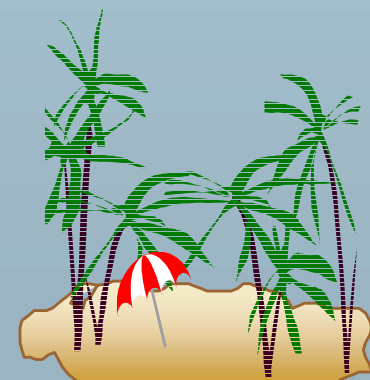
- 👉 Let $n(d)$ = number of terms in the document d
- 👉 $n(d, t)$ = number of occurrences of term t in the document d .
- 👉 Then relevance of a document d to a term t

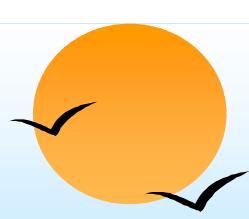
$$r(d, t) = \log \left(1 + \frac{n(d, t)}{n(d)} \right)$$

📄 The log factor is to avoid excessive weightage to frequent terms

👉 And relevance of document to query Q

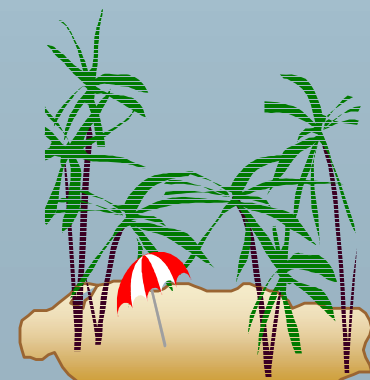
$$r(d, Q) = \sum_{t \in Q} \frac{r(d, t)}{n(t)}$$

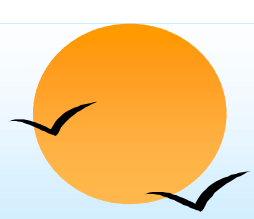




Relevance Ranking Using Terms (Cont.)

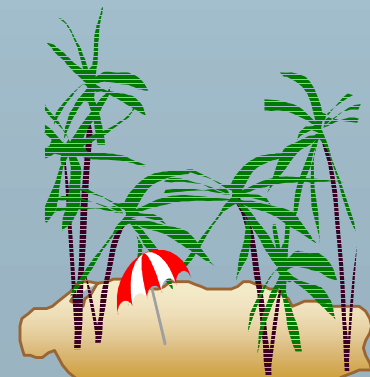
- Most systems add to the above model
 - ☞ Words that occur in title, author list, section headings, etc. are given greater importance
 - ☞ Words whose first occurrence is late in the document are given lower importance
 - ☞ Very common words such as “a”, “an”, “the”, “it” etc are eliminated
 - 📄 Called **stop words**
 - ☞ **Proximity**: if keywords in query occur close together in the document, the document has higher importance than if they occur far apart
- Documents are returned in decreasing order of relevance score
 - ☞ Usually only top few documents are returned, not all

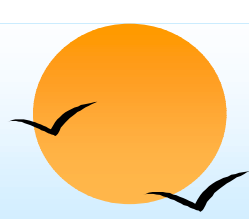




Relevance Using Hyperlinks

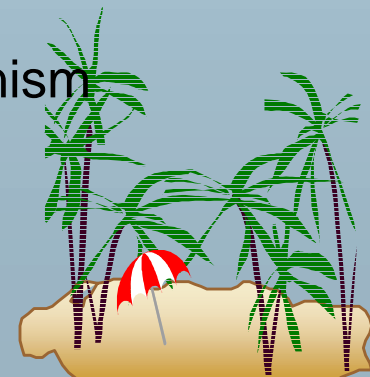
- When using keyword queries on the Web, the number of documents is enormous (many billions)
 - ☞ Number of documents relevant to a query can be enormous if only term frequencies are taken into account
- Using term frequencies makes “spamming” easy
 - 📄 E.g. a travel agent can add many occurrences of the words “travel agent” to his page to make its rank very high
- Most of the time people are looking for pages from popular sites
- Idea: use popularity of Web site (e.g. how many people visit it) to rank site pages that match given keywords
- Problem: hard to find actual popularity of site
 - ☞ Solution: next slide

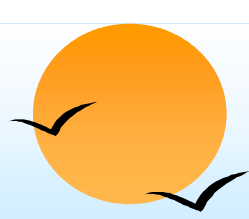




Relevance Using Hyperlinks (Cont.)

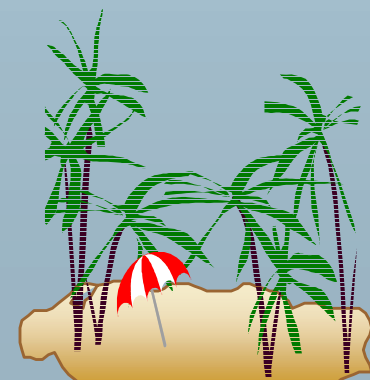
- Solution: use number of hyperlinks to a site as a measure of the popularity or **prestige** of the site
 - ☞ Count only one hyperlink from each site (why?)
 - ☞ Popularity measure is for site, not for individual page
 - ☞ Most hyperlinks are to root of site
 - ☞ Site-popularity computation is cheaper than page popularity computation
- Refinements
 - ☞ When computing prestige based on links to a site, give more weightage to links from sites that themselves have higher prestige
 - ☞ Definition is circular
 - ☞ Set up and solve system of simultaneous linear equations
 - ☞ Above idea is basis of the Google **PageRank** ranking mechanism

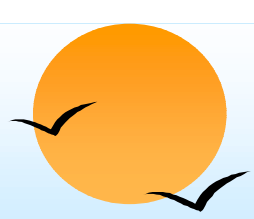




Relevance Using Hyperlinks (Cont.)

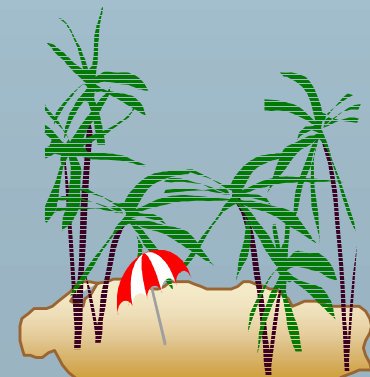
- Connections to **social networking** theories that ranked prestige of people
 - ☞ E.g. the president of the US has a high prestige since many people know him
 - ☞ Someone known by multiple prestigious people has high prestige
- Hub and authority based ranking
 - ☞ A **hub** is a page that stores links to many pages (on a topic)
 - ☞ An **authority** is a page that contains actual information on a topic
 - ☞ Each page gets a **hub prestige** based on prestige of authorities that it points to
 - ☞ Each page gets an **authority prestige** based on prestige of hubs that point to it
 - ☞ Again, prestige definitions are cyclic, and can be got by solving linear equations
 - ☞ Use authority prestige when ranking answers to a query

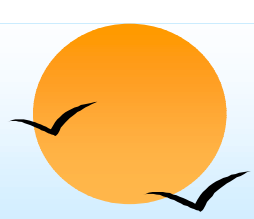




Similarity Based Retrieval

- Similarity based retrieval - retrieve documents similar to a given document
 - ☞ Similarity may be defined on the basis of common words
 - 📄 E.g. find k terms in A with highest $r(d, t)$ and use these terms to find relevance of other documents; each of the terms carries a weight of $r(d, t)$
- Similarity can be used to refine answer set to keyword query
 - ☞ User selects a few relevant documents from those retrieved by keyword query, and system finds other documents similar to these





Synonyms and Homonyms

■ Synonyms

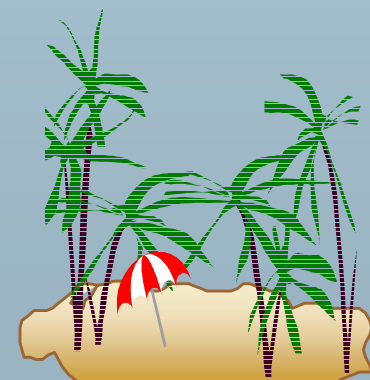
- 👉 E.g. document: “motorcycle repair”, query: “motorcycle maintenance”
 - 📄 need to realize that “maintenance” and “repair” are synonyms
- 👉 System can extend query as “motorcycle and (repair or maintenance)”

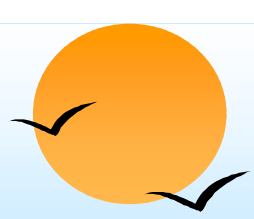
■ Homonyms

- 👉 E.g. “object” has different meanings as noun/verb
- 👉 Can disambiguate meanings (to some extent) from the context

■ Extending queries automatically using synonyms can be problematic

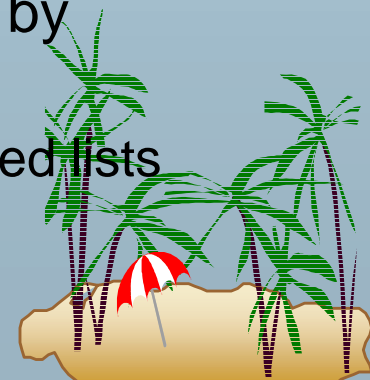
- 👉 Need to understand intended meaning in order to infer synonyms
 - 📄 Or verify synonyms with user
- 👉 Synonyms may have other meanings as well

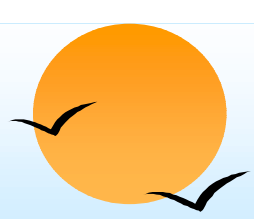




Indexing of Documents

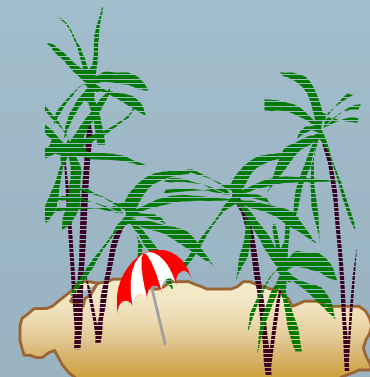
- An inverted index maps each keyword K_i to a set of documents S_i that contain the keyword
 - ☞ Documents identified by identifiers
- Inverted index may record
 - ☞ Keyword locations within document to allow proximity based ranking
 - ☞ Counts of number of occurrences of keyword to compute TF
- **and** operation: Finds documents that contain all of K_1, K_2, \dots, K_n .
 - ☞ Intersection $S_1 \cap S_2 \cap \dots \cap S_n$
- **or** operation: documents that contain at least one of K_1, K_2, \dots, K_n
 - ☞ union, $S_1 \cup S_2 \cup \dots \cup S_n$.
- Each S_i is kept sorted to allow efficient intersection/union by merging
 - ☞ “**not**” can also be efficiently implemented by merging of sorted lists

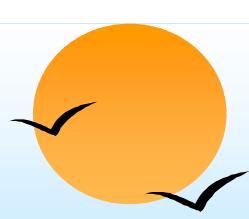




Measuring Retrieval Effectiveness

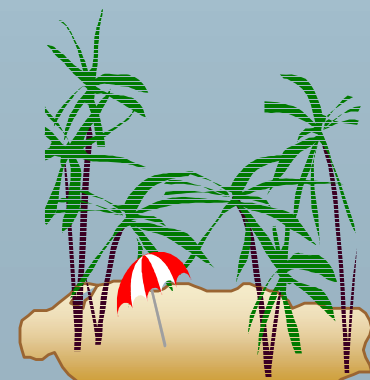
- IR systems save space by using index structures that support only approximate retrieval. May result in:
 - ☞ **false negative (false drop)** - some relevant documents may not be retrieved.
 - ☞ **false positive** - some irrelevant documents may be retrieved.
 - ☞ For many applications a good index should not permit any false drops, but may permit a few false positives.
- Relevant performance metrics:
 - ☞ **Precision** - what percentage of the retrieved documents are relevant to the query.
 - ☞ **Recall** - what percentage of the documents relevant to the query were retrieved.

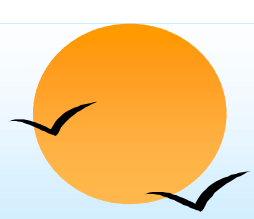




Measuring Retrieval Effectiveness (Cont.)

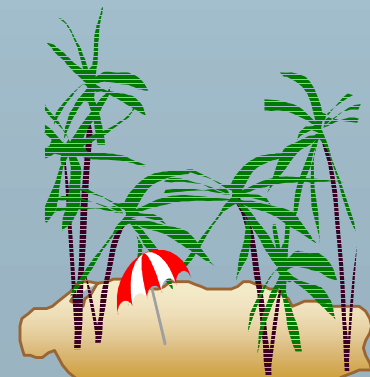
- Ranking order can also result in false positives/false negatives
 - ☞ Recall vs. precision tradeoff:
 - ☞ Can increase recall by retrieving many documents (down to a low level of relevance ranking), but many irrelevant documents would be fetched, reducing precision
 - ☞ Measures of retrieval effectiveness:
 - ☞ Recall as a function of number of documents fetched, or
 - ☞ **Precision as a function of recall**
 - Equivalently, as a function of number of documents fetched
 - ☞ E.g. “precision of 75% at recall of 50%, and 60% at a recall of 75%”
 - In general: draw a graph of precision vs recall.
- Problem: which documents are actually relevant, and which are not
 - ☞ Usual solution: human judges
 - ☞ Create a corpus of documents and queries, with humans deciding which documents are relevant to which queries
 - ☞ TREC (Text REtrieval Conference) Benchmark

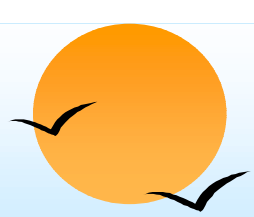




Web Crawling

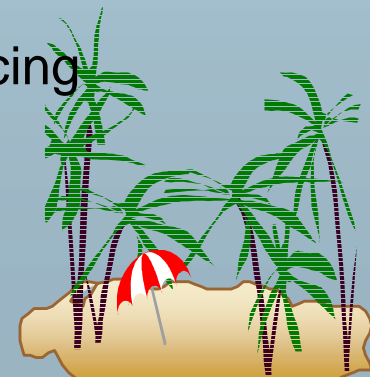
- **Web crawlers** are programs that locate and gather information on the Web
 - ☞ Recursively follow hyperlinks present in known documents, to find other documents
 - 📄 Starting from a *seed* set of documents
 - ☞ Fetched documents
 - 📄 Handed over to an indexing system
 - 📄 Can be discarded after indexing, or store as a *cached* copy
- Crawling the entire Web would take a very large amount of time
 - ☞ Search engines typically cover only a part of the Web, not all of it
 - ☞ Take months to perform a single crawl

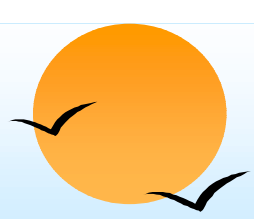




Web Crawling (Cont.)

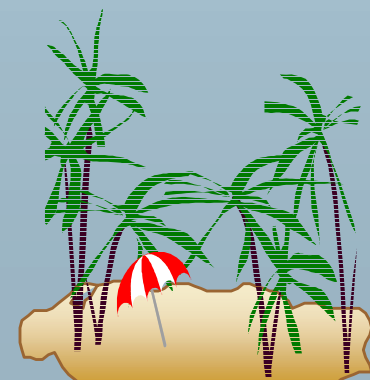
- Crawling is done by multiple processes on multiple machines, running in parallel
 - ☞ Set of links to be crawled stored in a database
 - ☞ New links found in crawled pages added to this set, to be crawled later
- Indexing process also runs on multiple machines
 - ☞ Creates a new copy of index instead of modifying old index
 - ☞ Old index is used to answer queries
 - ☞ After a crawl is “completed” new index becomes “old” index
- Multiple machines used to answer queries
 - ☞ Indices may be kept in memory
 - ☞ Queries may be routed to different machines for load balancing



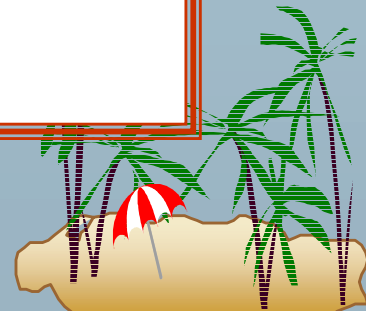
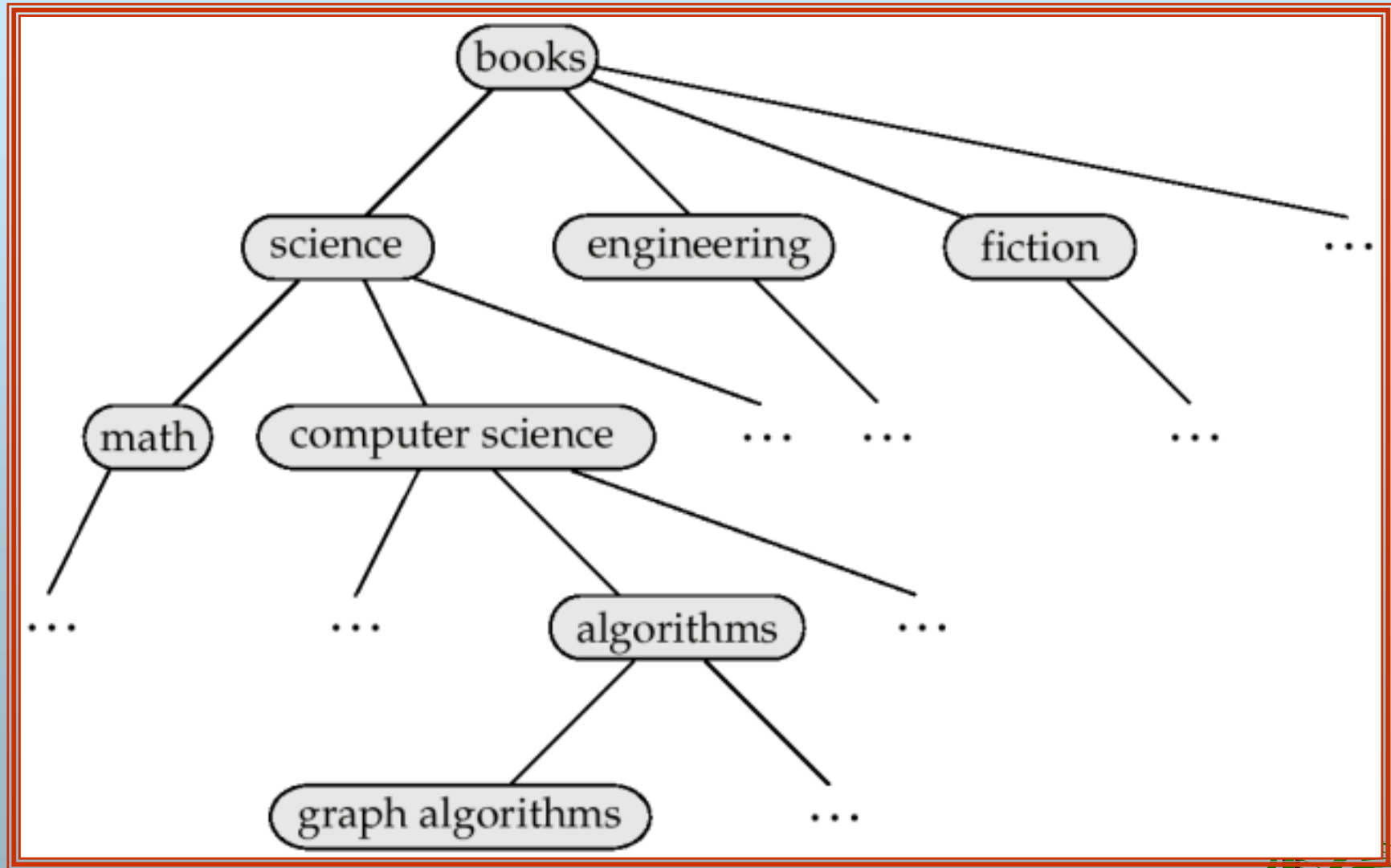


Browsing

- Storing related documents together in a library facilitates browsing
 - ☞ users can see not only requested document but also related ones.
- Browsing is facilitated by classification system that organizes logically related documents together.
- Organization is hierarchical: **classification hierarchy**



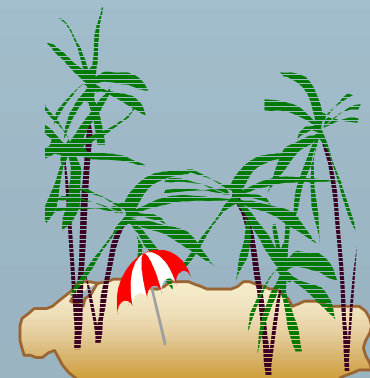
A Classification Hierarchy For A Library System

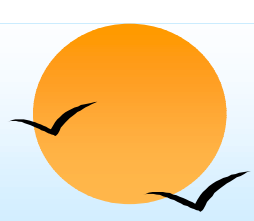




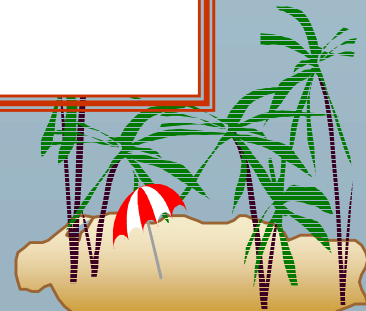
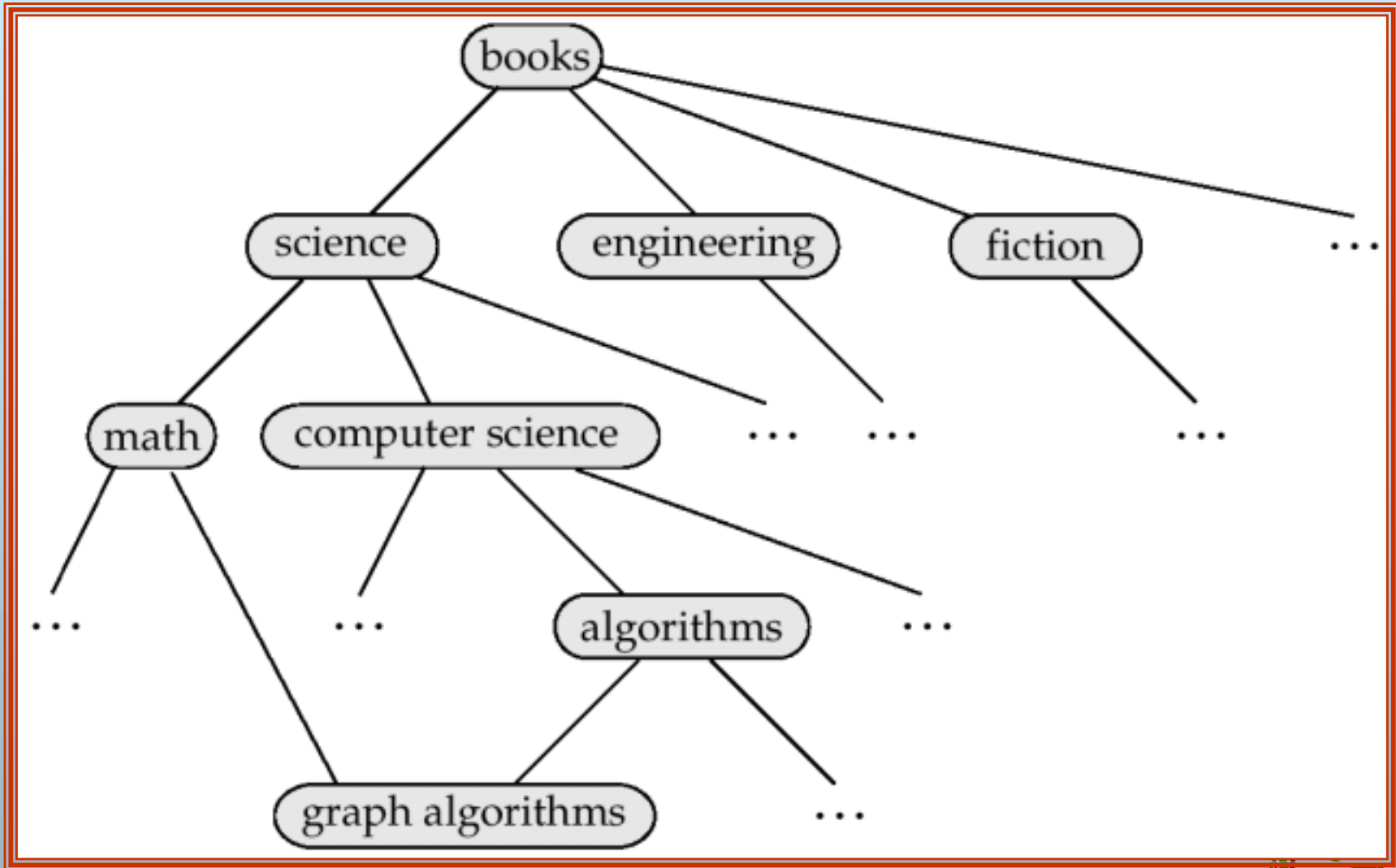
Classification DAG

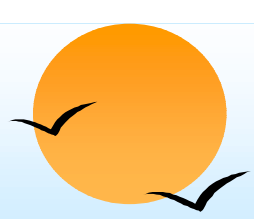
- Documents can reside in multiple places in a hierarchy in an information retrieval system, since physical location is not important.
- Classification hierarchy is thus Directed Acyclic Graph (DAG)





A Classification DAG For A Library Information Retrieval System





Web Directories

- A **Web directory** is just a classification directory on Web pages
 - ☞ E.g. Yahoo! Directory, Open Directory project
 - ☞ Issues:
 - ☞ What should the directory hierarchy be?
 - ☞ Given a document, which nodes of the directory are categories relevant to the document
 - ☞ Often done manually
 - ☞ Classification of documents into a hierarchy may be done based on term similarity

