# Chapter 3.5 Hash Structure of Multi-Dimensional Data

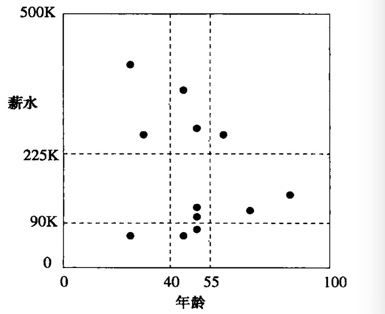
Build hash function using single key gets another two data structures:

* Grid File:Sort values of the dimension to divide this dimension.
* Piecewise Hash Function: Hash each dimensions and every dimension does influence the bucket.

## Chapter 3.5.1 Grid File

The simplest data structure compared with single index (hash function with single key) is Grind File. (The space be divided as the grind line space).

***Principles:***

* Grind line separates the dimension into strip, the dots on the grind line will be considered belongs to the strip which is lower than this grind line.
* Different grind can have different numbers of grind line. Interval length can be different within the neighboring grind lines.
* The same grind line can have different interval length.

***Example:***

Insert customer with two properties (age, salary) into two-dimension space:

|  |  |  |  |
| --- | --- | --- | --- |
| (25, 60) | (45, 60) | (50, 75) | (50, 100) |
| (50, 120) | (70, 110) | (85, 140) | (30, 260) |

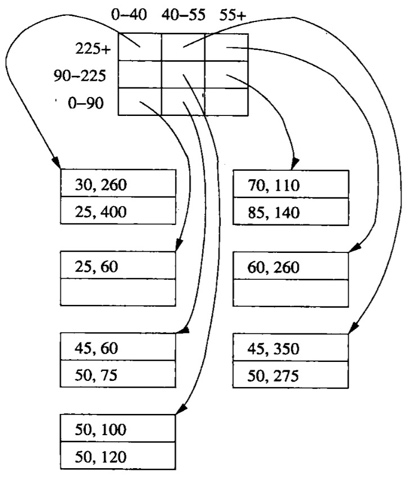
Look from the right image, twelve points are located in a single two-dimensional space. In every single dimensional, two lines are used to divide the current dimension into three parts. In sum, there are nine spaces in the current two-dimension space.

* 225K <= salary <= 500K, 0 <= age < 40;
* 225K <= salary <= 500K, 40 <= age < 55;
* 225K <= salary <= 500K, 55 <= age <= 100;
* 90K <= salary < 225K, 0 <= age < 40;
* 90K <= salary < 225K, 40 <= age < 55;
* 90K <= salary < 225K, 55 <= age <= 100;
* 0K <= salary < 90K, 0 <= age < 40;
* 0K <= salary < 90K, 40 <= age < 55;
* 0K <= salary < 90K, 55 <= age < 100;

## Chapter 3.5.2 Searching on Grid File

***Principles:***

* See every space as a bucket of hash table, and every single record in each space is seen as the record of this bucket.
* In order to locate a record, the value of every dimension should be confirmed. We focus on every record and its location in the grid space.
* The location of every single dimension decides its location in its bucket.

***Example:***

The bucket here is a 3\*3 matrix which is used to indicate nine spaces.

* Two empty spaces.
* Two records at most in every single bucket.
* Overflow blocks are un-necessary if no bucket has exceeded two records.

## Chapter 3.5.3 Insertion in Grid File

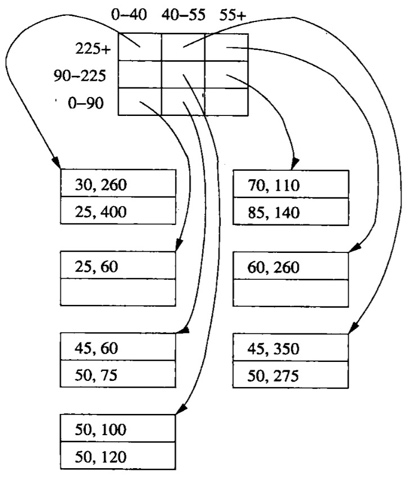
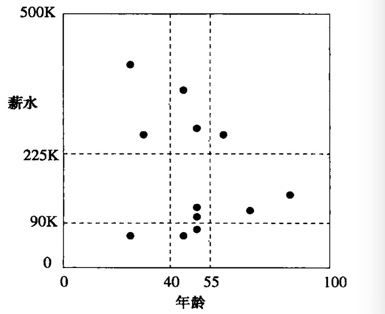
***Description:***

When record is needed to be inserted into a grid file, then we search the bucket and put the record into the bucket.

***Principle:***

* Find space in the bucket. -> Nothing has to be done.
* Can not find space in the bucket.
  + Add overflow blocks to the bucket.
  + Add or remove grid line to reconstruct structure.

Construct an index for the single dimension if the number of strip is very large. The searching key of index should be a set which is used to divide the current dimension.

***Example:***

Customer A: (52, 200K)

The record needs to be inserted into the middle of space. Read from the space, we can tell that there already have two records. The first one is (50, 100) and the second one is (50, 120).

The insertion of new customer (52, 200K) will exceed the limited value of bucket. Two solutions:

1. Add an overflow block for the bucket. The bucket will include three records, which contains:

(50, 100K), (50, 120K), (52, 200K)

1. Choose a grid line to spilt the bucket, either in age or salary dimension. Two records in one side while another one in another.

* Vertical line, such as age = 51. It separates the customer with the age 50 and 52.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | 0 – 40 | 40 – 51 | 51 – 55 | 55 + |
| 0k – 90k | (25, 60k) | (45, 60k)  (50, 75k) | Empty | Empty |
| 90k – 225k | Empty | (50, 100k)  (50, 120k) | (52, 200k) | (70, 110k)  (85, 140k) |
| 225k – 500k | (30, 260k)  (25, 400k) | (45, 350k)  (50, 275k) | Empty | (60, 260k) |

* Horizontal line, such as salary = 130k. It separates the customer with the salary 120k and 200k.

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 – 40 | 40 – 55 | 55 + |
| 0k – 90k | (25, 60k) | (45, 60k)  (50, 75k) | Empty |
| 90k – 130k | Empty | (50, 100k)  (50, 120k) | (70, 110k) |
| 130k – 225k | Empty | (52, 200k) | (85, 140k) |
| 225k – 500k | (30, 260k)  (25, 400k) | (45, 350k)  (50, 275k) | (60, 260k) |

* Horizontal line, such as salary = 115k. It separates the customer with the salary 100k and 120k, 110k and 140k.

|  |  |  |  |
| --- | --- | --- | --- |
|  | 0 – 40 | 40 – 55 | 55 + |
| 0k – 90k | (25, 60k) | (45, 60k)  (50, 75k) | Empty |
| 90k – 115k | Empty | (50, 100k) | (70, 110k) |
| 115k – 225k | Empty | (50, 120k)  (52, 200k) | (85, 140k) |
| 225k – 500k | (30, 260k)  (25, 400k) | (45, 350k)  (50, 275k) | (60, 260k) |

## Chapter 3.5.4 Grid File Performance

***Description:***

Consider the performance of searching from grid file.

***Principle:***

If data distribution is great in grid file, and the data file is not large, then we need to choose the grid line to make:

1. Less bucket and store the bucket into the main memory. So that when we search from bucket or insert a new line or add a new line will not cause disk I/O.

* Search specific record
* Search partial matching
* Search range matching
* Search the nearest neighbor

## Chapter 3.5.5 Piecewise Hash Function

## Chapter 3.5.6 Compare Between Grid File and Piecewise Hash Function