05_Dictionaries

September 22, 2019

1 5. Dictionaries

1.1 Basic operations on dictionaries

- Create dictionary:
  - key_list!value_list
  - NON-UNIQUE KEYS ARE TOLERATED
    * lookup and reverse lookup sees only firest occurrence
    * to get all keys to a given value: where 10=dict
  - NON-UNIFORM key and value types are tolerated
  - Both keys and values can be nested lists
  - To create a dictionary as a hash table (faster lookup speed): (`u#abc`)!(10;20;30)
    * Keys must be unique!!
  - singleton dictionary: (enlist "a")!(enlist 42)
  - empty dictionary: ()!()
  - typed empty dictionary:

[1]:
```
dict:(`symbol$())!`float$()
```

- Add item to dictionary:

[5]:
```
dict[`a]:44.0
```

[6]:
```
dict
```

[6]:
```
a| 44
```

- Assign new value to a key: dict["a"]:41
- UPSERT semantics: update and insert follows the same semantics
  - upsert semantics applies to tables and keyed tables (kdb+) as they consists of dictionaries
- Access / lookup:
  - single item: dict["a"] or dict "a"
  - multiple items: dict["a" "b"] / key can be a list variable
  - all keys: key dict
  - all values: value dict
  - If key not in dictionary, returns **null** whose type is that of the initial item in the value list.

  - find key by value: dict?10 -> returns the key of the first occurrence of 10 in a dictionary
    - results null if the value is not in the list
- Update item
- Remove item
  - single item: dict _ key
  - multiple items: list of keys _ dict or (enlist key) cut dict
    - removes all occurrences of duplicate keys!!!
- Length: count dict
- Transpose
- Other operations on dictionaries
  - join dictionaries:
    - d1+d2 values of the common items in the two dictionaries are added, otherwese union of items in the two domains are created
    - d1,d2: no addition, only union. if there are common elements in the domain, the right operand item prevails
      - not commutative: order matters
  - ^ (coalesce): dimilar to join (,), only non-null values prevail over null values
  - you can define dictionaries as lists if the keys are indices
    - create a sparse list by making non-continuus indexing in the dictionary's keys
  -

2 : extracting subdictionaries list hash dictionary: ("a";"b")#dict

- A dictionary is a mapping defined by an explicit association between a **key list and value list**.
- The two lists
  - must have the **same count** and
  - the key list should be a **unique collection**.
- While general lists can be used to create a dictionary, many dictionaries involve simple lists of keys.
- A dictionary is an **association between a list of keys and a list of values**.
- Logically it can also be considered as key-value pairs but it is **stored physically as a pair of lists**.
- NOTATION: **key__list!value__list**
- dictionary type: **99h**
- the order of keys and values in the list is significant:
  - differently ordered dictionaries are NOT identical: (`abc!10 20 30)~acb!10 30 20 -> 0b

[ ]:
```
eg1:(`Arthur`Dent; `Zaphod`Beeblebrox; `Ford`Prefect)! 100 42 150
eg2:1001 1002 1003!(`Arthur`Dent; `Zaphod`Beeblebrox; `Ford`Prefect)
```

[ ]:
```
typed_dict:(`symbol$())!`float$()
```

[ ]:
```
typed_dict[`a]:1f
```

[ ]:
```
typed_dict
```

[ ]:
```
eg2[1001 1002]
value eg2
```

[ ]:
```
d:`a`b`c!10 20 30
d[`a`c]
ks:`a`c
d ks
type d `a
d `x
```

2.1 5.2. Operations on dictionaries

[ ]:
```
.Q.s1 `a`b`c _ d
```

[ ]:
```
d:`a`b`c`a!10 20 30 11
```

[ ]:
```
d
```

[ ]:
```
(enlist `a) cut d
```

[ ]:
```
lst:1 2 3 1
```

[ ]:
```
lst except 1
```

2.1.1 5.2.4 Basic operations on maps

- arithmetic operations work on dictionaries the same as on lists: atom-wise

[ ]:
```
d1:`a`b`c!1 2 3
d2:`a`b`c!10 20 30
d1+d2
```

[ ]:
```
d1:`a`b`c!10 20 30
d2:`c`d!300 400
d1,d2
d2,d1
```

- Relational operations: For equality and comparison operations on dictionaries, the indicated operation is performed over the common keys. On disjoint keys, the appropriate null is effectively substituted for missing values.

2.2 5.3. Column dictionaries

- Definition: in a column dictionary, a rectangular list of lists are assigned to a list of symbols as keys
- The symbol is interpreted as a name, while the corresponding list as a vector of column values
- getting a field of a column dictionary: col_dict[name][index_of_field] or col_dict[name;index]
  - col_dict[;index]: retreives a record
- Transposed column dictionaries: flip col_dict = TABLE
- the result of the flipped col dict is just a logical adjustment for the sake of indexing order (row index first, colulmn index second)
  - meaning that the slots in indexing at depth are reversed.