INDEXING

CS 564- Fall 2018

ACKs: *Dan Suciu, Jignesh Patel, AnHai Doan*

WHAT IS THIS LECTURE ABOUT?

- Indexes
  - alternative file organization
- Index classifications:
  - hash vs tree
  - clustered vs unclustered
  - primary vs secondary

FILE ORGANIZATION: RECAP

- So far we have seen heap files
  - they store unordered data
  - fast for scanning all records in a file
  - fast for retrieving by record id (rid)

- But we also need alternative organizations of a file to support other access patterns

MOTIVATION

- Consider the following SQL query:

```
SELECT *
FROM Sales
WHERE Sales.date = “02-11-2016”
```

- For a heap file, we have to scan all the pages of the file to return the correct result

ALTERNATIVE FILE ORGANIZATIONS

- We can *speed up* the query execution by better organizing the data in a file
- There are many alternatives:
  - sorted files
  - indexes
    - B+ tree
    - hash index
    - bitmap index

INDEX BASICS

WHAT IS AN INDEX?

- **<u>Index</u>**: a data structure that organizes records of a table to optimize retrieval
  - it speeds up searches for a subset of records, based on values in certain (*search key*) attributes
  - any subset of the fields of a relation can be the search key
  - a search key is *not* the same as the primary key!
- An index contains a collection of *data entries* (each entry with enough info to locate the records)

HASH INDEX: EXAMPLE

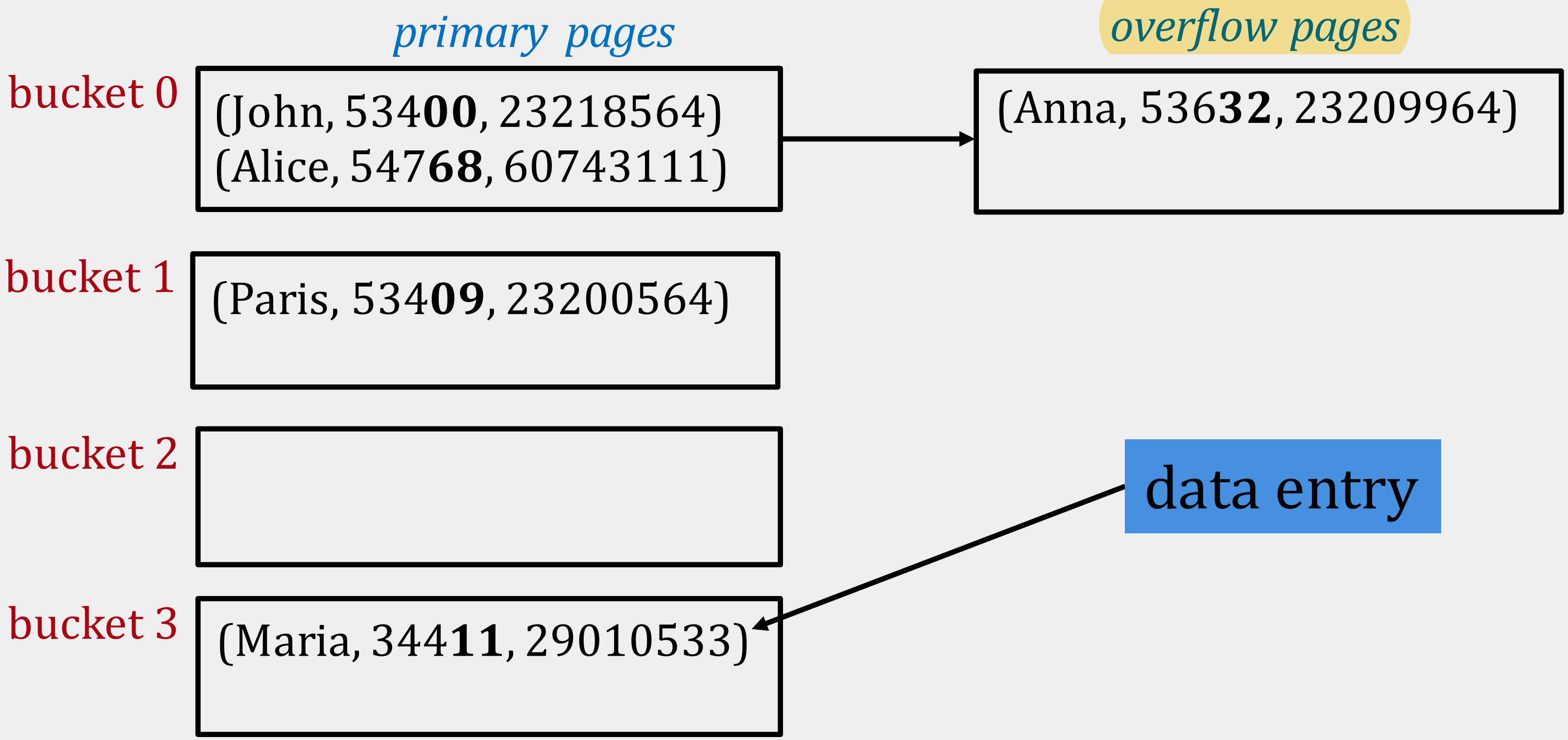
- A **hash index** is a collection of *buckets*
  - bucket = primary page + overflow pages
  - each bucket contains one or more data entries
- To find the bucket for each record, we use a hash function $\boldsymbol{h}$ applied on the search key k
  - $\boldsymbol{N}$ = number of buckets
  - $\boldsymbol{h}(k) \bmod \boldsymbol{N}$ = bucket in which the data entry belongs
- Records with different search key may belong in the same bucket!

HASH INDEX: EXAMPLE

`Person(name,zipcode,phone)`

- *search key*: zipcode
- *hash function* ***h***: last 2 digits

- 4 buckets
- each bucket has 2 data entries (full record)



DATA ENTRIES

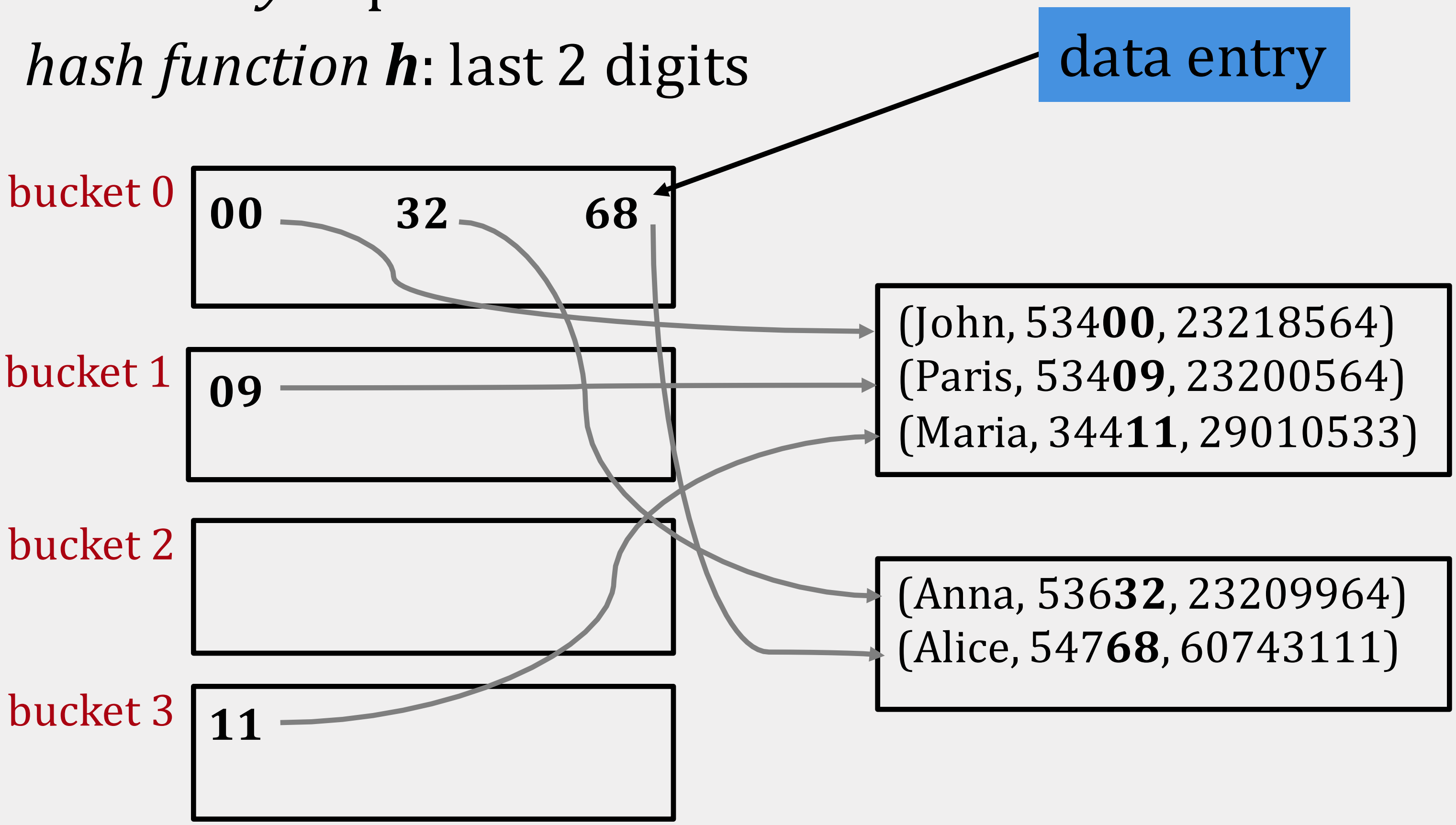
The actual data may not be in the same file as the index!

- In a data entry with search key **k** we have three alternatives of what to store:
  1. the record with key value **k**
  2. <**k**, rid of record with search key value **k**>
  3. <**k**, list of rids of records with search key **k**>
- The choice of alternative for data entries is independent of the indexing technique

EXAMPLE

Person(name,zipcode,phone)

- *search key*: zipcode
- *hash function* ***h***: last 2 digits



ALTERNATIVES FOR DATA ENTRIES

Alternative #1: *the data entry contains the record*

- the index structure is *by itself* a file organization for records
- *at most one* index on a given collection of data records should use alternative #1
- if data records are very large, the number of pages containing data entries is high
  - this means possibly slower search!

ALTERNATIVES FOR DATA ENTRIES

Alternatives #2, #3: *the data entry contains the rid*

- Data entries are typically much smaller than data records. So, better than #1 with large data records, especially if search keys are small
- #3 is more compact than #2, but leads to variable sized data entries even if search keys are of fixed length

MORE ON INDEXES

A file can have several indexes, on different search keys!

Index classification:

- *primary* **vs** *secondary*
- *clustered* **vs** *unclustered*

PRIMARY VS SECONDARY

- If the search key contains the primary key, it is called a primary index
  - in a primary index, there are no duplicates for a value of the search key
  - there can only be one primary index!
- Any other index is called a secondary index
- If the search key contains a candidate key, it is called a unique index
  - a unique index can also return no duplicates

EXAMPLE

`Sales (sid, product, date, price)`

1. An index on (sid) is a primary and unique index
2. An index on (date) is a secondary, but not unique, index

CLUSTERED INDEXES

Clustered index: the order of records matches the order of data entries in the index

- alternative #1 implies that the index is clustered
- a table can have at most one clustered index
- the cost of retrieving data records through the index varies greatly based on whether index is clustered or not

logical order of index ~ physical order of records

INDEXES IN PRACTICE

CHOOSING INDEXES

- What indexes should we create?
  - which relations should have indexes?
  - what field(s) should be the search key?
  - should we build several or one index?
- For each index, what kind of an index should it be?
  - clustered
  - hash/tree/bitmap

CHOOSING INDEXES

- Consider the best plan using the current indexes, and see if a better plan is possible with an additional index
- One must understand how a DBMS evaluates queries and creates query evaluation plans
- Important trade-offs:
  - queries go faster, updates are slower
  - more disk space is required

CHOOSING INDEXES

- Attributes in **WHERE** clause are candidates for index keys
  - exact match condition suggests hash index
  - indexes also speed up joins (later in class)
  - range query suggests tree index (B+ tree)

- Multi-attribute search keys should be considered when a **WHERE** clause contains several conditions
  - order of attributes is important for range queries
  - such indexes can enable index-only strategies for queries

COMPOSITE INDEXES

Composite search keys: search on a combination of fields (e.g. <date, price>)

- equality query: every field value is equal to a constant value
  - date="02-20-2015" and price =75
- range query: some field value is not a constant
  - date="02-20-2015"
  - date="02-20-2015" and price > 40

INDEXES IN SQL

```
CREATE INDEX index_name
ON table_name (column_name);
```

- Example of simple search key:

```
CREATE INDEX index1
ON Sales (price);
```

INDEXES IN SQL

```sql
CREATE UNIQUE INDEX index2
ON Sales (sid);
```

- A unique index does not allow any duplicate values to be inserted into the table
- It can be used to check efficiently integrity constraints (a duplicate value will not be allowed to be inserted)

INDEXES IN SQL

```
CREATE INDEX index3
ON Sales (date, price);
```

- Indexes with composite search keys are larger and more expensive to update
- They can be used if we have multiple selection conditions in our queries