Indexes

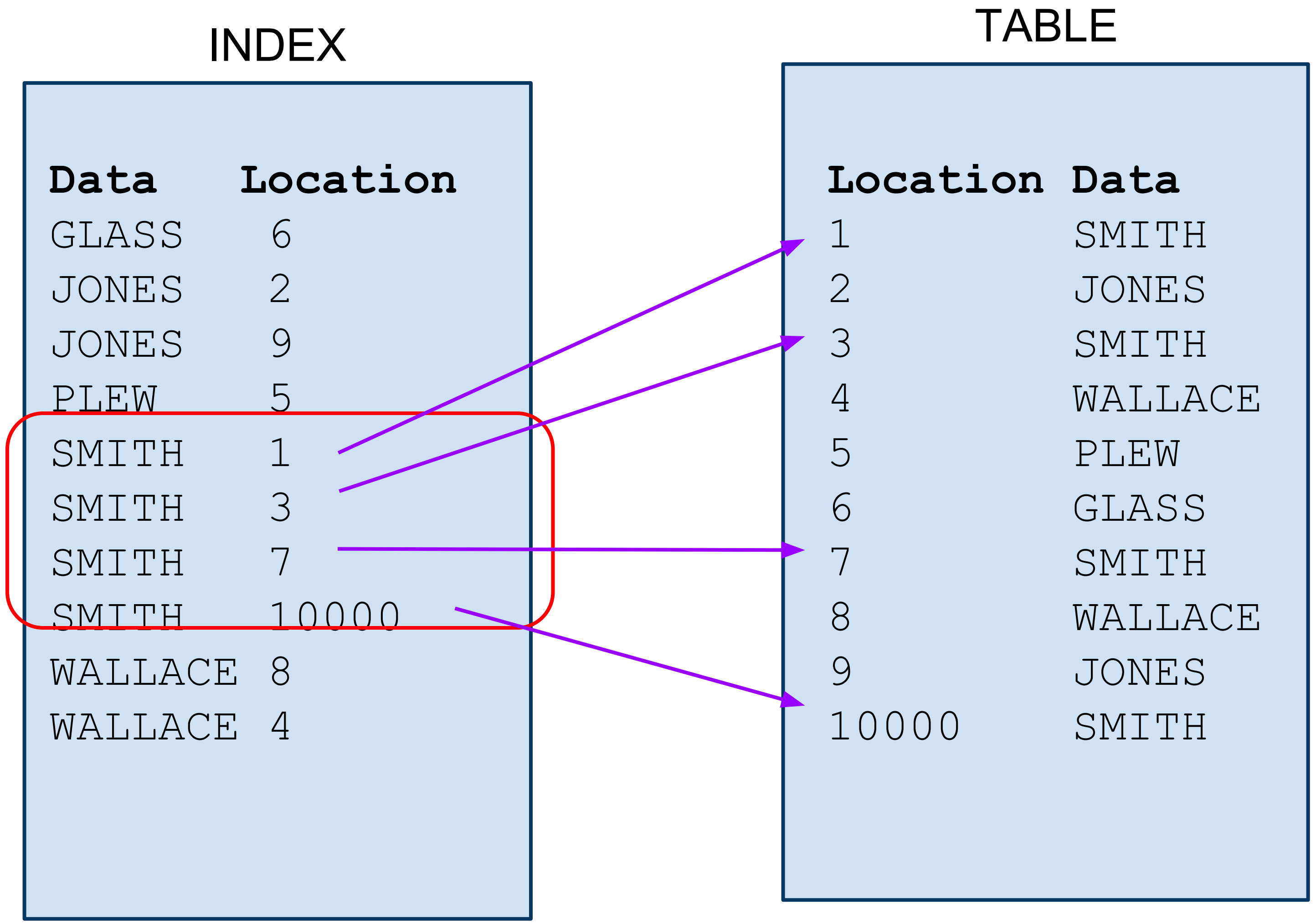
J. Porubän, M. Biňas, M. Nosáľ, D. Lakatoš (c) 2011 - 2018

Introduction

- **index** - persistent database object for easier locating of information
  - goal - increase data select performance
  - difference between going though all data and instantly finding wanting record
  - is created on column(s) of table
- you cannot choose to use index within your select - automatic process
  - Optimizer selects the best way to perform a query
- parallel in real world - index in book

How index works (hash table)

```
SELECT * FROM users
WHERE name='SMITH';
```



INDEX

Data	Location
GLASS	6
JONES	2
JONES	9
PLEW	5
SMITH	1
SMITH	3
SMITH	7
SMITH	10000
WALLACE	8
WALLACE	4

TABLE

Location	Data
1	SMITH
2	JONES
3	SMITH
4	WALLACE
5	PLEW
6	GLASS
7	SMITH
8	WALLACE
9	JONES
10000	SMITH

Index implementation

- **Balanced trees** (B-tree, B+ tree)
  - allows conditions =, <, > to be more effective
  - logarithmic complexity
- **Hash tables**
  - only =
  - almost constant complexity

- In Oracle pseudocolumn rowid uniquely identifies record in all DB and can be used as reference to finding record

Index properties

- create/drop of index does not corrupt or change DB data
- speeds up query operations, but slows down data modification operations
  - DML statements
  - index update
- their size may exceed the size of the table itself

Creating index

- syntax

```
CREATE INDEX index_name
ON table (col1 ASC|DESC,
col2 ASC|DESC,...)
```

- example

```
CREATE INDEX ixSubject
ON subject_student (id_subject);
CREATE INDEX ixNameSurname
ON student (name,surname);
```

Types of indexes I.

- **single-column indexes**
  - created on one column
  - simplest and most frequently used
  - most effective for queries with columns in `WHERE` clause
  - example: ids of records
  - syntax:

```
CREATE INDEX name
ON table (column)
```

Types of indexes II.

- **unique indexes**
    - used for performance and data integrity
    - do not allow duplicates in the table
    - example: passport number, SSN
        - primary key has this type of index
    - syntax:

```
CREATE UNIQUE INDEX name
ON table (column)
```

Types of indexes III.

- **composite indexes**
  - an index based on two or more columns of the table
  - order of columns is significant and has performance influence
    - starts with most important (most used) to the least
  - most effective when used on columns which are used in `WHERE` clause together
  - syntax:

```
CREATE INDEX name
ON table (col1, col2, ...)
```

Types of indexes IV.

- **implicit indexes**
  - are created automatically by DB system during object creation
  - example: primary key, unique constraint

Aspects determining the suitability of indices

- **size of table**
  - lower size, lower index performance gain
- **value distribution**
  - index helps find specific value (record)
- **load in form of selects vs. modifications**
  - selects are faster, modifications (inserts, updates, deletes) are slower
- the suitability of the index is always considered against **specific queries**

When to use index

- primary keys (automatic)
- foreign keys (most columns needed for joins)
- columns commonly used in `ORDER BY` or `GROUP BY`
- columns containing unique values
- columns used in `WHERE`, which return just small amount of records
- before deployment, it is best to test the functionality and performance of the indexes (test by experiment)

When to avoid indexes

- small tables
- tables, which are frequently modified
  - may be possible to solve by drop and recreate of index after data modification (still some performance drop)
- columns with a lot of `NULL` values
- columns, which are modified a lot
  - maintenance of the index can be challenging
- columns, which return a lot of records after filtering condition
  - e.g.: sex (gender)

Removing of index

- syntax

  ```
  DROP INDEX index_name
  ```
- be careful when removing index, as there may be a change in performance (increase/decrease)
- index can be once again recreated
  - without any data loss

Questions?