

# Advanced Database Systems

Spring 2026

## Lecture #16: Access Methods

R&G: Chapter 14

If you require this document in an alternative format, such as large print or a coloured background, please contact milos.nikolic@ed.ac.uk

1

## ACCESS METHODS

8

An **access method** (path) is a way the DBMS can access the data stored in a table

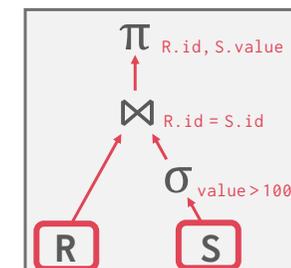
Not defined in relational algebra  
Includes selection **predicates**

Three basic approaches:

Sequential scan  
Index scan  
Multi-Index / "Bitmap" scan

Choice depends on #pages needed to read

```
SELECT R.id, S.value
FROM R, S
WHERE R.id = S.id
AND S.value > 100
```



8

## SEQUENTIAL SCAN

9

For each page in the table

Retrieve it from the buffer pool  
Iterate over each tuple and check if  
it matches (arbitrary) predicate  $p$

```
for page in table.pages:
  for t in page.tuples:
    if evalPred(p,t):
      // do something!
```

The DBMS keeps an internal **cursor** that tracks the last examined page

I/O cost = read  $N$  pages

Number of output pages =  $sel(p) \cdot N$  pages

$sel(p)$  - **selectivity** of predicate  $p$  is the fraction of tuples satisfying predicate  $p$   
The selection operator often processes tuples "on-the-fly" (no writing to disk)

9

## INDEX SCAN

10

The DBMS picks an index to find the tuples that the query needs

Which index to use depends on:

- What attributes the index contains
- What attributes the query references
- The attributes' value domains
- Predicate composition
- Whether the index has unique or non-unique keys
- Whether the index is clustered or unclustered
- ...

10

## INDEX SCAN

11

Suppose that a single table has two indexes

Tree index 1 on **age**

Index 2 on **dept**

```
SELECT * FROM Students
WHERE age < 30
AND dept = 'CS'
AND country = 'UK'
```

### Scenario #1

There are 99 people under the age of 30 but only 2 people in the CS department

### Scenario #2

There are 99 people in the CS department but only 2 people under the age of 30

11

## RECAP: INDEXES AND SELECTION

12

**Basic selection:** <key> <op> <constant>

Equality selections (*op* is =)

Range selections (*op* is one of <, >, <=, >=, BETWEEN)

B+ trees provide both

Hash indexes provide only equality

12

## RECAP: INDEXES AND ORDERING

13

Can index on any ordered subset of columns. Order matters!

Determines the selection predicates supported

In an ordered index (e.g., B+ tree), the keys are ordered **lexicographically** by the search key columns:

Ordered by the 1<sup>st</sup> column

2 entries match on 1<sup>st</sup> column? Ordered by 2<sup>nd</sup>

Match on 1<sup>st</sup> and 2<sup>nd</sup> column? Ordered by 3<sup>rd</sup>

...

ID	Name	Age	Salary
123	Jones	31	300
443	Smith	32	400
244	Gold	55	140
134	Alvaro	55	400
221	McDonald	79	300

Ordered lexicographically by the search key (Age, Salary)

13

## SEARCH KEY AND ORDERING

14

A tree index with a **composite search key** on columns ( $A_1, A_2, \dots, A_n$ ) "matches" a selection predicate if:

The predicate is a conjunction of  $m \geq 0$  equality clauses of the form:

$A_1 = c_1 \text{ AND } A_2 = c_2 \dots \text{ AND } A_m = c_m$

and at most 1 additional range clause of the form:

$\text{AND } A_{m+1} \text{ op } c_{m+1}$ , where *op* is one of <, >, <=, >=, BETWEEN

Why does this "match"? Lookup and scan in lexicographic order

Can do a lookup on equality conjuncts to find start-of-range

Can do a scan of contiguous data entries at leaves

Scan while  $A_{m+1} \text{ op } c_{m+1}$  holds. If no range clause, scan all matches to the first  $m$  conjuncts

14

## SEARCH KEY AND ORDERING

15

A tree index on (Age, Salary) matches which range predicates?

Legend

Green for rows we visit that are in the range

Red for rows we visit that are not in the range

ID	Name	Age	Salary
123	Jones	31	300
443	Smith	32	400
244	Gold	55	140
134	Alvaro	55	400
221	McDonald	79	300

15

## SEARCH KEY AND ORDERING

16

A tree index on (Age, Salary) matches which range predicates?

✓ Age = 31 and Salary = 400

ID	Name	Age	Salary
123	Jones	31	300
443	Smith	32	400
244	Gold	55	140
134	Alvaro	55	400
221	McDonald	79	300

16

## SEARCH KEY AND ORDERING

17

A tree index on (Age, Salary) matches which range predicates?

✓ Age = 31 and Salary = 400

✓ Age = 55 and Salary > 200

ID	Name	Age	Salary
123	Jones	31	300
443	Smith	32	400
244	Gold	55	140
134	Alvaro	55	400
221	McDonald	79	300

17

## SEARCH KEY AND ORDERING

18

A tree index on (Age, Salary) matches which range predicates?

✓ Age = 31 and Salary = 400

✓ Age = 55 and Salary > 200

✗ Age > 31 and Salary = 400

ID	Name	Age	Salary
123	Jones	31	300
443	Smith	32	400
244	Gold	55	140
134	Alvaro	55	400
221	McDonald	79	300

✗ Not a lexicographic range.  
Either visits useless rows or "bounce through" the index.

18

## SEARCH KEY AND ORDERING

19

A tree index on (Age, Salary) matches which range predicates?

- ✓ Age = 31 and Salary = 400
- ✓ Age = 55 and Salary > 200
- ✗ Age > 31 and Salary = 400
- ✓ Age = 31

ID	Name	Age	Salary
123	Jones	31	300
443	Smith	32	400
244	Gold	55	140
134	Alvaro	55	400
221	McDonald	79	300

✗ Not a lexicographic range.  
Either visits useless rows or "bounce through" the index.

19

## SEARCH KEY AND ORDERING

20

A tree index on (Age, Salary) matches which range predicates?

- ✓ Age = 31 and Salary = 400
- ✓ Age = 55 and Salary > 200
- ✗ Age > 31 and Salary = 400
- ✓ Age = 31
- ✓ Age > 31

ID	Name	Age	Salary
123	Jones	31	300
443	Smith	32	400
244	Gold	55	140
134	Alvaro	55	400
221	McDonald	79	300

✗ Not a lexicographic range.  
Either visits useless rows or "bounce through" the index.

20

## SEARCH KEY AND ORDERING

21

A tree index on (Age, Salary) matches which range predicates?

- ✓ Age = 31 and Salary = 400
- ✓ Age = 55 and Salary > 200
- ✗ Age > 31 and Salary = 400
- ✓ Age = 31
- ✓ Age > 31
- ✗ Salary = 300

ID	Name	Age	Salary
123	Jones	31	300
443	Smith	32	400
244	Gold	55	140
134	Alvaro	55	400
221	McDonald	79	300

✗ Not a lexicographic range.  
Either visits useless rows or "bounce through" the index.

21

## INDEX-ONLY SCAN

22

### Index-only plans

Queries might be answered without retrieving any tuples from one or more of the table if a suitable index is available

### Index-only scans

Retrieve only matching search keys from index pages, without reading data pages

Often much faster than heap scans due to small index sizes

```
SELECT E.dno, COUNT(*)
FROM Employee E
GROUP BY E.dno
```

Index on E.dno

```
SELECT E.dno, MIN(E.salary)
FROM Employee E
GROUP BY E.dno
```

Tree index on (E.dno, E.salary)

```
SELECT AVG(E.salary)
FROM Employee E
WHERE E.age = 25
AND E.salary > 300
```

Tree index on (E.age, E.salary)

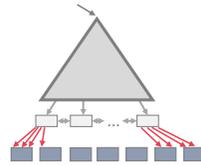
22

## CLUSTERED B+ TREE SCAN

23

A **clustered B+ tree** index whose search key matches the selection predicate  $p$  is clearly the superior method

I/O cost =  $2-4$  + (to reach a leaf page)  
 $sel(p) \cdot (\# \text{ of leaf pages})$  (to scan leaf pages)



If variant **B** or **C**, we may also need to access data records

Requires reading  $sel(p) \cdot (\# \text{ of data pages})$  pages

But if the query uses only search key attributes, then **no need to access data records!**

23

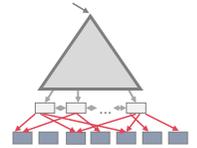
## UNCLUSTERED B+ TREE SCAN

24

Accessing an unclustered B+ tree index can be expensive

I/O cost  $\approx$  # of matching **leaf index entries**

But index-only scans as fast as with clustered B+ trees!



If  $sel(p)$  indicates a large number of qualifying records, it pays off to

read the matching index entries  $\langle k, rid \rangle$

sort those entries on their  $rid$  field

access the pages in sorted  $rid$  order

Lack of clustering is a minor issue if  $sel(p)$  is close to 0

24

## HASH INDEX SCAN

25

A hash index matches a selection predicate  $p$  only if:

- 1)  $p$  contains a term of the form  $A = c$ , and
- 2) the hash index has been built over column  $A$

Composite search keys must be bounded entirely

A hash index on  $(age, dept)$  matches  $age = 27$  AND  $dept = 'CS'$

But does **not** match  $age = 27$

Use index to jump to the bucket of qualifying tuples

Scan pages in that bucket looking for matches

If search key values are unique, terminate after finding a match

Otherwise, scan all pages in that bucket

25

## MULTI-INDEX SCAN

26

If there are multiple indexes that the DBMS can use for a query:

Compute sets of record IDs using each matching index

Combine these sets based on the query's predicates (union vs. intersect)

Retrieve the records and apply any remaining terms

Set intersection can be done with bitmaps, hash tables, or Bloom filters

Postgres calls this Bitmap Scan

26

## MULTI-INDEX SCAN

Suppose that a single table has two indexes

Tree Index 1 on **age**

Index 2 on **dept**

```
SELECT * FROM Students
WHERE age < 30
AND dept = 'CS'
AND country = 'UK'
```

DBMS may decide to use both indexes

Retrieve the record ids satisfying **age < 30** using Tree Index 1

Retrieve the record ids satisfying **dept = 'CS'** using Index 2

Take their intersection

Retrieve records and check **country = 'UK'**

## PRACTICAL EXERCISE WITH POSTGRESQL

### Bay Area Bike Sharing dataset

Download from Learn → Practice Worksheets

Import into PostgreSQL

Follow instructions from Learn → Practice Worksheets → Lecture16 – Practical Exercise

### Understanding query plans

How PostgreSQL executes a query

Interpreting EXPLAIN and EXPLAIN ANALYZE output

### Choice of access method

Index scan vs sequential scan vs bitmap scan

Depends on table size, row count, and selectivity

Planner picks method based on estimated cost

## PRACTICAL EXERCISE WITH POSTGRESQL (2)

### Bitmap Index Scan

Uses an index to locate candidate rows

Builds a bitmap of matching row positions

Efficient for queries matching many rows

### Bitmap Heap Scan

Fetches actual rows from the table using the bitmap

Rechecks conditions for correctness

Reduces random I/O by reading table pages in batches