Contents

1 Problem 3: File Organization and Indexing (GQ3)

1.1 Problem Description

Given the relational database table:

VeterinaryClinic (vet_name, license_no, clinic_city, fee_per_visit)

The following insertions are performed on the table VeterinaryClinic:

1. Insert record <Smith, 12, Tulsa, $30>
2. Insert record <Brown, 45, OKC, $25>
3. Insert record <Wilson, 23, Norman, $20>
4. Insert record <Taylor, 78, OKC, $25>
5. Insert record <Davis, 34, Edmond, $30>
6. Insert record <Clark, 67, Enid, $35>
7. Insert record <Lewis, 89, OKC, $25>
8. Insert record <Walker, 56, Yukon, $30>
9. Insert record <Harris, 90, Tulsa, $35>

Assumptions:

- Each block can store up to **3 veterinarian records**
- VeterinaryClinic is organized as a **sequential file** with **vet_name** as the ordering field

1.2 Tasks

1. Show the contents of the file after the last insertion
2. Show the contents of the dense primary index and secondary index on fee_per_visit (assuming index-sequential file)
3. Show the content of the B+-tree index file on license_no with order 3

2 Part 1: Sequential File Contents After Last Insertion

2.1 Sorted Records by vet_name

When organizing as a sequential file ordered by vet_name, we first sort all records alphabetically:

Position	vet_name	license_no	clinic_city	fee_per_visit
1	Brown	45	OKC	$25
2	Clark	67	Enid	$35
3	Davis	34	Edmond	$30
4	Harris	90	Tulsa	$35
5	Lewis	89	OKC	$25
6	Smith	12	Tulsa	$30
7	Taylor	78	OKC	$25
8	Walker	56	Yukon	$30
9	Wilson	23	Norman	$20

Table 1: Records Sorted by vet_name

2.2 Block Organization (3 records per block)

BLOCK 0 (Address: 0x000)

Record 0 (0x000.0): <Brown, 45, OKC, $25>
Record 1 (0x000.1): <Clark, 67, Enid, $35>
Record 2 (0x000.2): <Davis, 34, Edmond, $30>

BLOCK 1 (Address: 0x001)

Record 0 (0x001.0): <Harris, 90, Tulsa, $35>
Record 1 (0x001.1): <Lewis, 89, OKC, $25>
Record 2 (0x001.2): <Smith, 12, Tulsa, $30>

BLOCK 2 (Address: 0x002)

Record 0 (0x002.0): <Taylor, 78, OKC, $25>
Record 1 (0x002.1): <Walker, 56, Yukon, $30>
Record 2 (0x002.2): <Wilson, 23, Norman, $20>

2.3 Detailed Address Mapping

Record Address	Block	Position	Data
0x000.0	Block 0	Pos 0	<Brown, 45, OKC, $25>
0x000.1	Block 0	Pos 1	<Clark, 67, Enid, $35>
0x000.2	Block 0	Pos 2	<Davis, 34, Edmond, $30>
0x001.0	Block 1	Pos 0	<Harris, 90, Tulsa, $35>
0x001.1	Block 1	Pos 1	<Lewis, 89, OKC, $25>
0x001.2	Block 1	Pos 2	<Smith, 12, Tulsa, $30>
0x002.0	Block 2	Pos 0	<Taylor, 78, OKC, $25>
0x002.1	Block 2	Pos 1	<Walker, 56, Yukon, $30>
0x002.2	Block 2	Pos 2	<Wilson, 23, Norman, $20>

Table 2: Complete Address Mapping

3 Part 2: Index-Sequential File with Indexes

3.1 Dense Primary Index on vet_name

A **dense primary index** has one index entry for **every search key value** in the data file.

Search Key (vet_name)	**Record Pointer (Address)**
Brown	0x000.0
Clark	0x000.1
Davis	0x000.2
Harris	0x001.0
Lewis	0x001.1
Smith	0x001.2
Taylor	0x002.0
Walker	0x002.1
Wilson	0x002.2

Table 3: Dense Primary Index on vet_name

Explanation:

- Each veterinarian name has an entry pointing to its exact record location
- Total entries: 9 (one for each record)
- This is a **primary index** because vet_name is the ordering field
- This is **dense** because every search key value has an index entry

3.2 Secondary Index on fee_per_visit

A **secondary index** is built on a non-ordering field. For fee_per_visit, multiple records may have the same value, so we use a structure with record lists.

Search Key (fee_per_visit)	**Record Pointers (Addresses)**
$20	0x002.2 (Wilson)
$25	0x000.0 (Brown) 0x001.1 (Lewis) 0x002.0 (Taylor)
$30	0x000.2 (Davis) 0x001.2 (Smith) 0x002.1 (Walker)
$35	0x000.1 (Clark) 0x001.0 (Harris)

Table 4: Secondary Index on fee_per_visit

Explanation:

- This is a **secondary index** because fee_per_visit is NOT the ordering field
- Each unique fee value points to ALL records with that fee
- The index is sorted by fee_per_visit for efficient searching
- Total unique entries: 4 ($20, $25, $30, $35)

4 Part 3: B+-Tree Index on license_no (Order 3)

4.1 B+-Tree Properties

For a B+-tree of **order n = 3**:

- **Maximum keys per node:** n - 1 = 2 keys
- **Minimum keys per internal node:** $\lceil n/2 \rceil$ - 1 = $\lceil 1.5 \rceil$ - 1 = 1 key
- **Minimum keys per leaf node:** $\lceil n/2 \rceil$ - 1 = 1 key
- **Maximum children per internal node:** n = 3
- **Minimum children per internal node:** $\lceil n/2 \rceil$ = 2

4.2 License Numbers in Sorted Order

The license numbers in sorted order are: 12, 23, 34, 45, 56, 67, 78, 89, 90

4.3 Step-by-Step B+-Tree Construction

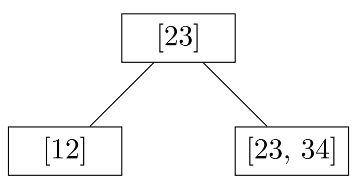
4.3.1 Insertions 1-2: Insert 12, 23

After inserting 12 and 23:

[12, 23]

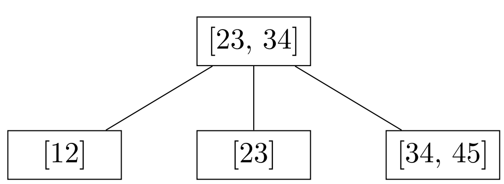
4.3.2 Insertion 3: Insert 34 (causes split)

Node becomes [12, 23, 34] which exceeds maximum of 2 keys. Split and promote middle key (23):



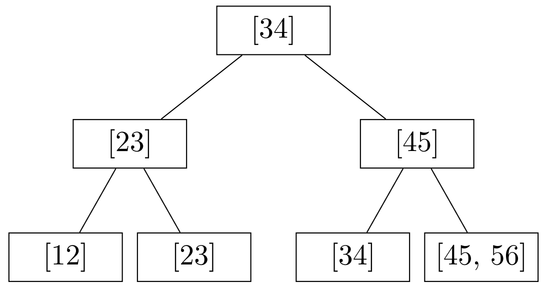
4.3.3 Insertion 4: Insert 45 (causes split)

Leaf [23, 34] becomes [23, 34, 45]. Split and promote 34:



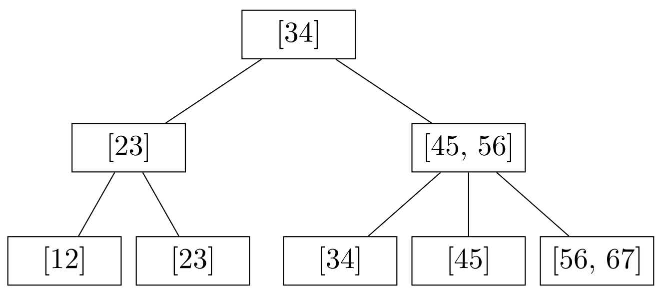
4.3.4 Insertion 5: Insert 56 (causes split and root split)

Leaf [34, 45] becomes [34, 45, 56]. Split and promote 45. Root [23, 34] becomes [23, 34, 45], which is full. Split root:



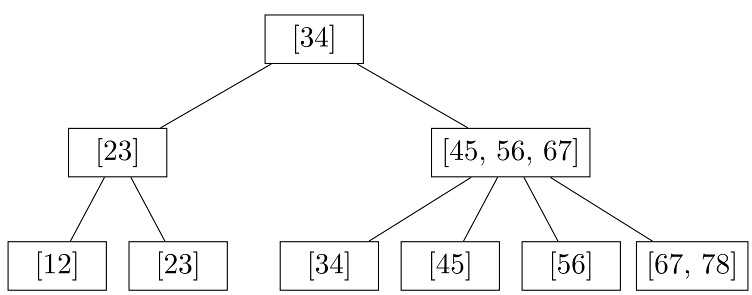
4.3.5 Insertion 6: Insert 67 (causes split)

Leaf [45, 56] becomes [45, 56, 67]. Split and promote 56:

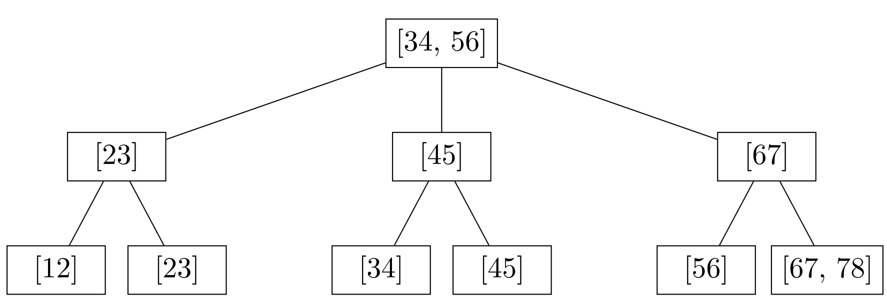


4.3.6 Insertion 7: Insert 78 (causes split)

Leaf [56, 67] becomes [56, 67, 78]. Split and promote 67:

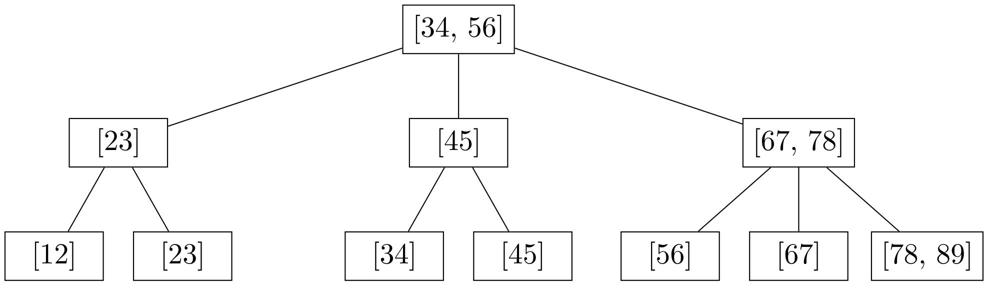


Internal node [45, 56, 67] has 3 keys (exceeds max of 2). Split and promote 56 to root:



4.3.7 Insertion 8: Insert 89 (causes split)

Leaf [67, 78] becomes [67, 78, 89]. Split and promote 78:



4.3.8 Insertion 9: Insert 90 (causes split and root split)

Leaf [78, 89] becomes [78, 89, 90]. Split and promote 89. Internal node [67, 78] becomes [67, 78, 89], which exceeds max. Split and promote 78 to root. Root [34, 56] becomes [34, 56, 78], exceeds max. Split root and promote 56:

4.4 Final B+-Tree Structure

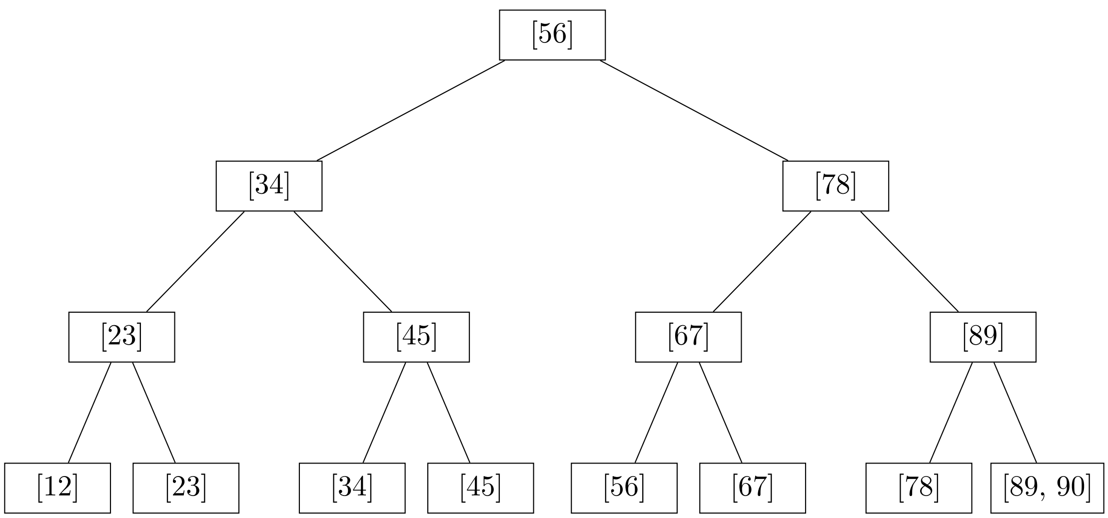


Figure 1: Final B+-Tree on license_no

4.5 Leaf Node Contents with Data Pointers

The leaf nodes contain the following license numbers with pointers to the actual veterinarian records:

License No	vet_name	Leaf Node
12	Smith	[12]
23	Wilson	[23]
34	Davis	[34]
45	Brown	[45]
56	Walker	[56]
67	Clark	[67]
78	Taylor	[78]
89	Lewis	[89, 90]
90	Harris	[89, 90]

Table 5: Leaf Node to Data Record Mapping

Note: In a B+-tree, leaf nodes are typically linked horizontally (not shown in diagram) to support efficient range queries.

4.6 B+-Tree Properties Verification

- **Root has between 2 and 3 children:** Root [56] has 2 children - Yes
- **Internal nodes have between 2 and 3 children:** All internal nodes have 2 children - Yes
- **All leaves are at the same level:** Yes, depth = 3 - Correct
- **Leaf nodes have between 1 and 2 keys:** All leaves comply - Yes
- **Keys are in ascending order:** Yes - Correct
- **No duplicate license_no:** Correct (unique constraint) - Yes

5 Summary

This solution demonstrates three file organization and indexing methods for the VeterinaryClinic database:

- **Part 1:** Sequential file organization with 9 records sorted by vet_name and stored in 3 blocks (3 records per block)
- **Part 2:** Index-sequential file with:
  - Dense primary index on vet_name (9 entries)
  - Secondary index on fee_per_visit (4 unique fee values: $20, $25, $30, $35)
- **Part 3:** B+-tree of order 3 on license_no with final structure having root [56], demonstrating step-by-step insertion and splits

All three approaches provide different trade-offs between storage overhead, search efficiency, and maintenance complexity, with sequential files being simplest, index-sequential adding fast lookup capability, and B+-trees offering the best performance for dynamic data with frequent updates.