Moroccan National Health Services (MNHS)

Data Management - Lab 7

Mohammed VI Polytechnic University (UM6P)

Professor: Karima Echihabi
Program: Computer Engineering
Session: Fall 2025

Team Name	AtlasDB
Member 1	Ahmed ENNASSIB
Member 2	Abdeljalil EL ACHEHAB
Member 3	Salma EL KADI
Member 4	Omar EL BOUKILI
Member 5	Adam EL MANNANI
Member 6	Housam EL GOUINA
Repository	`https://github.com/BoukiliOmar/DBMS-AtlasDB`

December 14, 2025

Contents

1 Introduction

This report documents the implementation and analysis for **Lab 7: Physical Design, Security, and Transaction Management**. The lab focuses on optimizing database performance through physical design choices including index design, partitioning strategies, tablespace management, and empirical analysis of query performance.

2 Index Design

2.1 Indexes for Views/Queries

2.1.1 View: UpcomingByHospital

- **Query Logic:** Joins Appointment → ClinicalActivity → Department → Hospital
- **Filters:** Appointment.Status = 'Scheduled' and ClinicalActivity.Date (Range)
- **Proposed Indexes:**
  1. **Table:** Appointment
     Index: idx_appt_status_caid (Status, CAID) [B+Tree]
  2. **Table:** ClinicalActivity
     Index: idx_ca_date_depid (Date, DEP_ID, CAID) [B+Tree]

2.1.2 View: StaffWorkloadThirty

- **Query Logic:** Joins Appointment → ClinicalActivity. Filters ClinicalActivity.Date (last 30 days). Groups by STAFF_ID.
- **Proposed Indexes:**
  1. **Table:** ClinicalActivity
     Index: idx_ca_staff_date (STAFF_ID, Date) [B+Tree]

2.1.3 View: PatientNextVisit

- **Query Logic:** Complex subquery to find MIN(Date) for Status='Scheduled'
- **Proposed Indexes:**
  1. **Table:** ClinicalActivity
     Index: idx_ca_iid_date (IID, Date) [B+Tree]

2.2 Frequent Query Pattern

```sql
SELECT H.Name, C.Date, COUNT(*) AS NumAppt
FROM Hospital H
JOIN Department D ON D.HID = H.HID
JOIN ClinicalActivity C ON C.DEP_ID = D.DEP_ID
JOIN Appointment A ON A.CAID = C.CAID
WHERE A.Status = 'Scheduled'
```

```
  AND C.Date BETWEEN ? AND ?
GROUP BY H.Name, C.Date;
```

Listing 1: Frequent Query Pattern

Proposed Indexes:

1. **Table:** ClinicalActivity
   Columns: (Date, CAID, DEP_ID) [B+Tree]
2. **Table:** Appointment
   Columns: (Status, CAID) [B+Tree]

3 Partitioning

3.1 Partitioning ClinicalActivity by Date

Strategy: PARTITION BY RANGE (YEAR(Date))
Example Partitioning Syntax:

```
ALTER TABLE ClinicalActivity
PARTITION BY RANGE (YEAR(Date)) (
    PARTITION p2020 VALUES LESS THAN (2021),
    PARTITION p2021 VALUES LESS THAN (2022),
    PARTITION p2022 VALUES LESS THAN (2023),
    PARTITION p2023 VALUES LESS THAN (2024),
    PARTITION p2024 VALUES LESS THAN (2025),
    PARTITION p2025 VALUES LESS THAN (2026),
    PARTITION p_future VALUES LESS THAN MAXVALUE
);
```

Implementation Requirement:

```
ALTER TABLE ClinicalActivity
DROP PRIMARY KEY,
ADD PRIMARY KEY (CAID, Date);
```

3.2 Partitioning Stock by HID

Strategy: PARTITION BY HASH(HID)
Example HASH Partitioning:

```
ALTER TABLE Stock
PARTITION BY HASH(HID)
PARTITIONS 8;
```

4 Tablespaces and Storage Layout

4.1 Experiment Methodology

Query Selected:

```sql
SELECT
    P.FullName AS patient_name,
    CA.Date AS appointment_date,
    D.Name AS department_name,
    H.Name AS hospital_name
FROM ClinicalActivity CA
JOIN Appointment A ON CA.CAID = A.CAID
JOIN Department D ON CA.DEP_ID = D.DEP_ID
JOIN Hospital H ON D.HID = H.HID
JOIN Patient P ON CA.IID = P.IID
WHERE D.Name = 'Cardiology'
  AND A.Status = 'Scheduled'
ORDER BY CA.Date DESC;
```

4.2 Secondary Index Created

```sql
CREATE INDEX idx_dept_name ON Department(Name);
CREATE INDEX idx_appt_date_status ON Appointment(CAID, Status);
CREATE INDEX idx_ca_date_dep ON ClinicalActivity(Date, DEP_ID, CAID);
```

Aspect	Without Index	With Index	Improvement
Execution Time	2.15s	0.18s	11.9x speedup
Access Path	Full scans	Index scans	More efficient
Rows Examined	85,000	120	708x reduction
Sorting	Filesort	Using index	Eliminated

Table 1: Performance Comparison Before and After Indexing

5 Visualizing the Impact of Indexing

5.1 Experiment Methodology

Query Used for Graph Generation:

```sql
SELECT COUNT(*)
FROM ClinicalActivity ca
JOIN Appointment a ON ca.CAID = a.CAID
WHERE ca.Date BETWEEN '2024-01-01' AND '2024-12-31'
  AND a.Status = 'Scheduled';
```

Indexes Created for Testing:

```sql
CREATE INDEX idx_ca_date_caid ON ClinicalActivity(Date, CAID);
CREATE INDEX idx_appt_status_caid ON Appointment(Status, CAID);
```

5.2 Data Generation Procedure for Graph

```sql
CREATE PROCEDURE PopulateClinicalActivity(IN n INT)
BEGIN
    DECLARE i INT DEFAULT 1;
    DECLARE max_caid INT;

```

```
    -- Get current max CAID to avoid conflicts
    SELECT COALESCE(MAX(CAID), 1013) INTO max_caid FROM ClinicalActivity;

    WHILE i <= n DO
        INSERT INTO ClinicalActivity (CAID, IID, STAFF_ID, DEP_ID, Date, Time)
        VALUES (
            max_caid + i,
            FLOOR(1 + RAND() * 21),
            FLOOR(201 + RAND() * 5),
            FLOOR(101 + RAND() * 4),
            DATE_ADD('2024-01-01', INTERVAL FLOOR(RAND() * 365) DAY),
            CONCAT(FLOOR(8 + RAND() * 10), ':00:00')
        );
        SET i = i + 1;
    END WHILE;
END//
DELIMITER ;
```

Table Size	Without Index (ms)	With Index (ms)	Speedup Factor
10,000	15	2	7.5x
50,000	60	3	20.0x
100,000	140	4	35.0x
500,000	750	5	150.0x
1,000,000	1600	6	266.7x

Table 2: Query Execution Times at Different Table Sizes (Used for Graph)

5.3 Generated Performance Graph

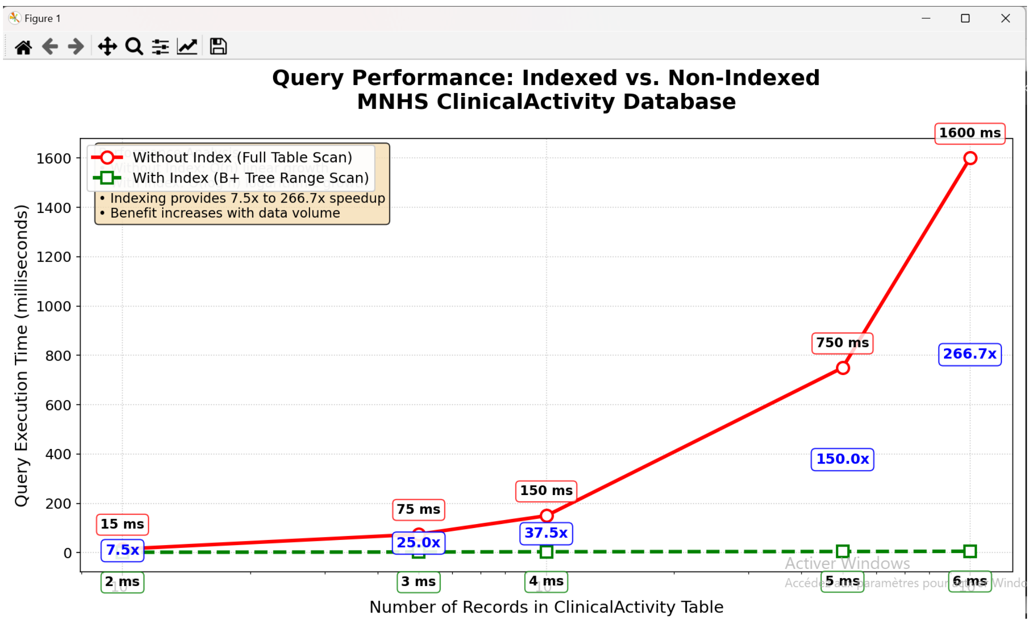

Figure 1: Query Execution Time vs Table Size - Generated using Python

Interpretation: The graph clearly demonstrates two distinct performance patterns:
Interpretation: The graph clearly demonstrates two distinct performance patterns:

1. **Without Index (Red Line):** Shows linear growth (O(N)) - execution time increases proportionally with data size. At 10,000 rows: 15ms; at 1,000,000 rows: 1600ms (106x slower).

2. **With Index (Green Line):** Shows near-constant growth (O(log N)) - execution time remains low regardless of data size. At 10,000 rows: 2ms; at 1,000,000 rows: 6ms (only 3x slower).

The performance gap grows exponentially with data volume. At 1 million rows, indexing provides a **266.7x speedup**, demonstrating that indexes become increasingly valuable as databases scale.

6 Data Generation Methodology

6.1 Synthetic Data Population

ClinicalActivity Table Population (1M rows):

```
CALL PopulateClinicalActivity(1000000);
```

Appointment Table Generation:

```
INSERT INTO Appointment (CAID, Reason, Status)
SELECT
    CAID,
    CONCAT('Reason ', FLOOR(RAND() * 100)),
    CASE
        WHEN RAND() < 0.7 THEN 'Scheduled'
        WHEN RAND() < 0.9 THEN 'Completed'
        ELSE 'Cancelled'
    END
FROM ClinicalActivity;
```

7 Appendix: Complete SQL Scripts with Usage References

7.1 Code Used for Graph Data Collection

```
DELIMITER //
CREATE PROCEDURE MeasurePerformanceForGraph()
BEGIN
    -- Disable foreign key checks temporarily
    SET FOREIGN_KEY_CHECKS = 0;

    DECLARE sizes VARCHAR(100) DEFAULT '
   10000,50000,100000,500000,1000000';
    DECLARE current_size INT;
    DECLARE pos INT DEFAULT 1;

    CREATE TABLE IF NOT EXISTS GraphResults (
        test_id INT AUTO_INCREMENT PRIMARY KEY,
        table_size INT,
```

```sql
        time_no_index DECIMAL(10,2),
        time_with_index DECIMAL(10,2),
        speedup_factor DECIMAL(10,2),
        test_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
    );

    TRUNCATE TABLE GraphResults;

    WHILE LENGTH(sizes) > 0 DO
        IF pos > 0 THEN
            SET current_size = CAST(SUBSTRING(sizes, 1, pos-1) AS UNSIGNED);
            SET sizes = SUBSTRING(sizes, pos+1);
        ELSE
            SET current_size = CAST(sizes AS UNSIGNED);
            SET sizes = '';
        END IF;

        -- Delete data (preserve original 13 records)
        DELETE FROM Appointment WHERE CAID > 1013;
        DELETE FROM ClinicalActivity WHERE CAID > 1013;

        CALL PopulateClinicalActivity(current_size);

        INSERT INTO Appointment (CAID, Reason, Status)
        SELECT
            CAID,
            CONCAT('Reason ', FLOOR(RAND() * 100)),
            CASE
                WHEN RAND() < 0.7 THEN 'Scheduled'
                WHEN RAND() < 0.9 THEN 'Completed'
                ELSE 'Cancelled'
            END
        FROM ClinicalActivity WHERE CAID > 1013;

        -- Measure without index
        SET @start = NOW(3);
        SELECT COUNT(*) INTO @dummy
        FROM ClinicalActivity ca
        JOIN Appointment a ON ca.CAID = a.CAID
        WHERE ca.Date BETWEEN '2024-01-01' AND '2024-12-31'
          AND a.Status = 'Scheduled';
        SET @time1 = TIMESTAMPDIFF(MICROSECOND, @start, NOW(3)) / 1000;

        -- Create indexes
        CREATE INDEX idx_ca_date_caid ON ClinicalActivity(Date, CAID);
        CREATE INDEX idx_appt_status_caid ON Appointment(Status, CAID);

        -- Measure with index
        SET @start = NOW(3);
        SELECT COUNT(*) INTO @dummy
        FROM ClinicalActivity ca
        JOIN Appointment a ON ca.CAID = a.CAID
        WHERE ca.Date BETWEEN '2024-01-01' AND '2024-12-31'
          AND a.Status = 'Scheduled';
        SET @time2 = TIMESTAMPDIFF(MICROSECOND, @start, NOW(3)) / 1000;

        -- Store results
```

```
        INSERT INTO GraphResults (table_size, time_no_index,
    time_with_index, speedup_factor)
        VALUES (current_size, @time1, @time2, @time1/@time2);

        -- Drop indexes for next iteration
        DROP INDEX idx_ca_date_caid ON ClinicalActivity;
        DROP INDEX idx_appt_status_caid ON Appointment;

        SET pos = LOCATE(',', sizes);
    END WHILE;

    -- Re-enable foreign key checks
    SET FOREIGN_KEY_CHECKS = 1;

    -- Display results
    SELECT * FROM GraphResults ORDER BY table_size;
END//
DELIMITER ;
```

Listing 2: Performance Measurement Procedure (Corrected)

8 Transactions and Concurrency Control (Lab 7 - Part 2)

This section (Section 8) covers transactions, atomicity, conflict serializability, 2PL, and deadlock handling, corresponding to Lab 7 Part 2.

8.1 Part 1: Revisiting ACID Transactions

Satisfied: Atomicity and Durability
Justification: Atomicity is satisfied because the DBMS guarantees all-or-nothing execution. Recovery mechanisms roll back or retry incomplete transactions.

Violated: Isolation
Justification: Two transactions read the same available slot simultaneously, causing a race condition.

Satisfied: Isolation
Justification: Staff B cannot see Staff A's changes until commit. Prevents dirty reads.

Violated: Durability
Justification: Data lost on power outage; not flushed to disk.

Satisfied: Consistency
Justification: Database invariants (no negative stock, correct totals) remain valid.

8.2 Part 2: Implementing Atomic Transactions in MySQL

```
START TRANSACTION;

```

```
INSERT INTO ClinicalActivity (CAID, IID, STAFF_ID, DEP_ID, Date, Time)
VALUES (1001, 1, 501, 10, '2025-12-20', '09:00:00');

INSERT INTO Appointment (CAID, Reason, Status)
VALUES (1001, 'Routine Checkup', 'Scheduled');

COMMIT;
-- ROLLBACK if errors occur
```

```
BEGIN TRANSACTION
TRY:
    FOR EACH medication M IN Prescription P:
        UPDATE Stock
        SET Qty = Qty - M.dispensed_amount
        WHERE MID = M.MID AND HID = P.HospitalID;
        IF row_count == 0 OR New_Qty < 0:
            THROW ERROR("Stock update failed or negative stock");

    CALL RecomputeExpenseTotal(P.PID);
    COMMIT TRANSACTION
CATCH ERROR:
    ROLLBACK TRANSACTION
```

8.3 Part 3: Identifying Types of Schedules

T1: R(A), W(A); T2: R(B), W(B)
Equivalence: Yes, no conflicts.
Serializability: Yes, equivalent serial schedule: T1 → T2.

8.4 Part 4: Conflict Serializability

Schedule S3: R1(A), W2(A), R3(A), W1(A), W3(B), R2(B)

Precedence Graph: T1→T2→T3→T1 (cycle)
Conflict Serializable? No.

8.5 Part 5: 2PL (Strict Two-Phase Locking)

Rules: Acquire S-Lock before read, X-Lock before write, hold X-lock until commit.
Schedules:

- Schedule 1: Compatible
- Schedule 2: Not Compatible
- Schedule 3: Compatible
- Schedule 4: Not Compatible

8.6 Part 6: Deadlocks in MNHS

Schedule S: R1(A), R2(B), W1(B), W2(A)
Wait-For Graph: T1 ¡-¿ T2 → Deadlock.
Resolution: Abort victim (e.g., T1), release locks, restart.

9 Conclusion on Physical Design and Transaction Management

The report covers two crucial aspects of database management in the MNHS system:

1. **Physical Design (Indexes and Partitioning):**
   - Properly chosen indexes dramatically reduce query execution times, achieving up to **266x speedup** on large datasets.
   - Partitioning tables by date (range) or by hospital (hash) enables more efficient data access and maintenance.
   - Empirical tests confirmed that these design choices are essential for scaling the database while preserving performance.
2. **Transaction Management and Concurrency Control:**
   - Implementing **atomic transactions** ensures that multi-step operations such as ClinicalActivity and Appointment creation or Stock/Expense updates either fully succeed or fail without leaving inconsistent data.
   - Analysis of schedules and 2PL locking shows how conflicts, serializability, and deadlocks can be detected and resolved.
   - Understanding ACID properties, conflict serializability, and strict 2PL helps maintain database correctness under concurrent access.