

UNIT 7

Developing Stored Procedures, DML Triggers and Indexing

(5 Hours) — Database Management System (CACS255) · BCA II Year / IV Semester

⚠ Important Note — Please Read

These notes were prepared quickly from the course syllabus and standard textbook knowledge. **They may contain errors, typing mistakes or incomplete points.** Please take care: cross-check definitions, formulas, SQL syntax and examples with your class notes, the prescribed textbook and your teacher before relying on them for exams or practicals.

7.1 Managing Stored Procedures

A **stored procedure** is a named, pre-compiled group of SQL statements stored in the database and executed by calling its name. It may accept input parameters, return output values and contain programming logic (variables, IF, WHILE, TRY...CATCH). Examples here use T-SQL.

- **Faster** — the execution plan is compiled and cached.
- **Reusable** and easy to maintain — logic is kept in one place.
- **Secure** — users can be given EXECUTE permission without direct access to tables; helps prevent SQL injection.
- **Less network traffic** — one call instead of many statements.

Create, Alter, Drop and Execute

Basic stored procedure

```
CREATE PROCEDURE GetAllStudents
AS
BEGIN
    SELECT * FROM Student;
END;
GO
EXEC GetAllStudents;           -- execute (EXECUTE or EXEC)
ALTER PROCEDURE GetAllStudents -- change the definition
AS
BEGIN
    SELECT StudentID, Name FROM Student ORDER BY Name;
END;
GO
DROP PROCEDURE GetAllStudents; -- remove
```

*Note: MySQL syntax: **DELIMITER //** **CREATE PROCEDURE** name() **BEGIN ... END //**
DELIMITER ; and **CALL** name();*

Passing Data to Stored Procedures — Parameters

Input parameter

```
CREATE PROCEDURE GetStudentsByDept
    @DeptID INT                -- input parameter
AS
BEGIN
    SELECT * FROM Student WHERE DeptID = @DeptID;
END;
GO
EXEC GetStudentsByDept @DeptID = 10;    -- named
EXEC GetStudentsByDept 10;              -- positional
```

Parameter Default Values

Parameters with defaults

```
CREATE PROCEDURE GetStudents
    @DeptID INT = NULL,        -- default = NULL
    @Country VARCHAR(30) = 'Nepal' -- default = 'Nepal'
AS
BEGIN
    SELECT * FROM Student
    WHERE (@DeptID IS NULL OR DeptID = @DeptID) AND Country = @Country;
END;
GO
EXEC GetStudents;              -- uses both defaults
EXEC GetStudents @DeptID = 20; -- overrides one
```

Returning Data — OUTPUT Parameter and RETURN Statement

Method	What it returns	Notes
OUTPUT parameter	Any data type — one or more values	Declare with OUTPUT in both the procedure and the call.
RETURN statement	Only an integer status code	0 usually means success; non-zero means error.
Result set	Rows produced by SELECT	Read by the caller as a table.

OUTPUT parameter and RETURN

```
CREATE PROCEDURE CountStudents
    @DeptID INT,
    @Total INT OUTPUT                -- output parameter
AS
BEGIN
    SELECT @Total = COUNT(*) FROM Student WHERE DeptID = @DeptID;
    RETURN 0;                       -- status code
END;
GO
DECLARE @cnt INT, @status INT;
EXEC @status = CountStudents @DeptID = 10, @Total = @cnt OUTPUT;
PRINT 'Total students = ' + CAST(@cnt AS VARCHAR(10));
```

Encryption of a Stored Procedure

WITH ENCRYPTION

```
CREATE PROCEDURE SecretReport
WITH ENCRYPTION                    -- definition text is hidden
AS
    SELECT Name, Salary FROM Employee;
```

With **WITH ENCRYPTION** the source text of the procedure is obfuscated so that users cannot read it (for example through *sp_helptext*). Keep a copy of the source, because it cannot be easily recovered from the database.

Transaction Flow inside Procedures

A transaction groups statements so that either all succeed or none do. In T-SQL: **BEGIN TRANSACTION** → statements → **COMMIT** (save) or **ROLLBACK** (undo).

Transaction flow with TRY...CATCH

```
CREATE PROCEDURE TransferMoney @From INT, @To INT, @Amt DECIMAL(10,2)
AS
BEGIN
    BEGIN TRY
        BEGIN TRANSACTION;
        UPDATE Account SET Balance = Balance - @Amt WHERE AccNo = @From;
        UPDATE Account SET Balance = Balance + @Amt WHERE AccNo = @To;
        COMMIT TRANSACTION;                -- both updates succeed
    END TRY
    BEGIN CATCH
        ROLLBACK TRANSACTION;              -- any error undoes everything
    END CATCH
END;
```

7.2 DML Triggers

A **trigger** is a special stored procedure that **runs automatically** when an INSERT, UPDATE or DELETE event happens on a table. It cannot be called directly and takes no parameters. Uses: auditing, enforcing complex business rules, maintaining derived data.

Type	When it fires
AFTER (FOR) trigger	After the triggering statement has completed and constraints have been checked.
INSTEAD OF trigger	In place of the triggering statement (often used on views).

Inside a trigger two special temporary tables are available:

Table	INSERT	UPDATE	DELETE
inserted	New rows	New (after) values	Empty
deleted	Empty	Old (before) values	Removed rows

Audit trigger (creating a trigger)

```
CREATE TRIGGER trg_AuditDelete
ON Student
AFTER DELETE
AS
BEGIN
    INSERT INTO StudentAudit (StudentID, Name, DeletedOn)
    SELECT StudentID, Name, GETDATE() FROM deleted;    -- handles many rows
END;
```

Developing Multi-Row Enabled Triggers

A single INSERT/UPDATE/DELETE statement can affect **many rows**, but a trigger fires **only once per statement**. A trigger written for one row (e.g. *SELECT @id = StudentID FROM inserted*) silently ignores the other rows. Therefore triggers must be **set-based**: use joins, INSERT...SELECT, or check @@ROWCOUNT and the whole inserted/deleted tables.

Multi-row safe trigger

```
CREATE TRIGGER trg_CheckSalary ON Employee AFTER INSERT, UPDATE
AS
BEGIN
    IF EXISTS (SELECT 1 FROM inserted WHERE Salary < 0)    -- checks ALL rows
    BEGIN
        RAISERROR('Salary cannot be negative', 16, 1);
        ROLLBACK TRANSACTION;
    END
END;
```

Trigger Limitations

- Triggers hide logic — hard to debug and can slow down data modifications.
- They fire once per statement, not once per row (in SQL Server); careless code breaks with multi-row changes.
- Cannot be created on temporary tables or system tables; cannot take parameters or return values.
- Can cause **cascading/recursive** triggers if a trigger modifies another table that has a trigger.
- Some statements, e.g. TRUNCATE TABLE, do not fire DELETE triggers.

Altering, Disabling, Enabling and Dropping a Trigger

Managing triggers

```
DISABLE TRIGGER trg_AuditDelete ON Student;    -- stop it firing
ENABLE  TRIGGER trg_AuditDelete ON Student;    -- start again
DISABLE TRIGGER ALL ON Student;                -- all triggers of the table
ALTER TABLE Student DISABLE TRIGGER trg_AuditDelete; -- alternative form
DROP TRIGGER trg_AuditDelete;                  -- delete permanently
```

7.3 Indexing

Basic Concept of Indexing

An **index** is a separate data structure that speeds up searching, like the index at the end of a book. It stores **search-key values** with **pointers** to the actual records. Without an index the DBMS must scan the whole table (*full table scan*); with an index it can jump straight to the required rows. The cost: extra storage and slower INSERT/UPDATE/DELETE, because indexes must be maintained.

Ordered Indices

In an **ordered index** the index entries are stored **sorted** by the search-key value.

Type	Description
Primary (clustered) index	Index on the attribute by which the file is physically sorted. Only one per table. Usually on the primary key.
Secondary (non-clustered) index	Index on a non-ordering attribute; the file is not sorted by it. Many allowed per table.
Dense index	One index entry for every search-key value / record.
Sparse index	Index entries for only some values (one per block); needs a sorted file.

Types of Indexing

- **Single-level index** — one index file; can become large.
- **Multi-level index** — an index on the index, reducing the number of disk accesses.
- **B+ tree index** — a balanced multi-level tree; the default in most RDBMSs. Supports equality and range searches and stays balanced after inserts and deletes.
- **Hash index** — a hash function maps the key to a bucket; excellent for equality search, useless for range search.
- **Unique index** — does not allow duplicate values.
- **Composite index** — built on two or more columns.
- **Bitmap index** — bit vectors for columns with few distinct values (e.g. gender).

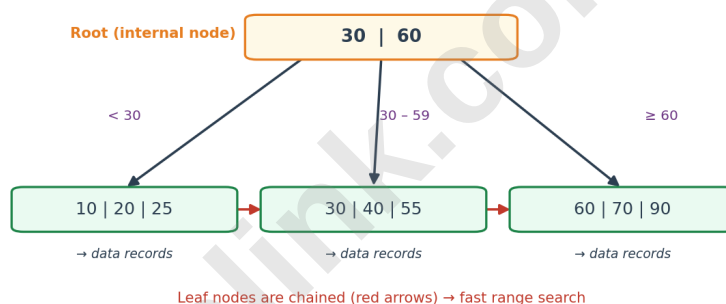


Figure 7.1 — A simple B+ tree index (root and linked leaf nodes)

Multiple Key Access

Queries often have conditions on several attributes, e.g. *WHERE DeptID = 20 AND Salary = 60000*. Options are:

- Use the index on one attribute and test the other condition on the fetched records;
- Use **index intersection** — search both indexes and intersect the two sets of record pointers;
- Create a **composite (multiple-key) index** on (DeptID, Salary). It works best when the query uses the **left-most** columns of the index first.

Creating and Dropping an Index

Index statements

```
CREATE INDEX idx_student_name ON Student (Name);           -- simple
CREATE UNIQUE INDEX idx_student_email ON Student (Email);  -- unique
CREATE INDEX idx_emp_dept_sal ON Employee (DeptID, Salary DESC); -- composite
CREATE CLUSTERED INDEX idx_cl ON Student (StudentID);      -- T-SQL clustered
CREATE NONCLUSTERED INDEX idx_nc ON Student (Name);         -- T-SQL non-clustered
DROP INDEX idx_student_name ON Student;                    -- T-SQL and MySQL
-- Oracle / PostgreSQL: DROP INDEX idx_student_name;
```

Note: A PRIMARY KEY or UNIQUE constraint automatically creates an index. Do not index every column — index columns used often in WHERE, JOIN and ORDER BY.

Key Takeaways — Unit 7

- ✓ A stored procedure is a precompiled, reusable SQL routine: CREATE, ALTER, DROP, EXEC; supports parameters, defaults, OUTPUT and RETURN.
- ✓ RETURN gives only an integer status; OUTPUT parameters give any value; WITH ENCRYPTION hides the source.
- ✓ Transactions use BEGIN TRANSACTION, COMMIT and ROLLBACK (with TRY...CATCH for errors).
- ✓ Triggers fire automatically on INSERT/UPDATE/DELETE and use the inserted and deleted tables; write them to handle multiple rows.
- ✓ DISABLE/ENABLE TRIGGER controls a trigger; DROP TRIGGER removes it.
- ✓ Indexes speed up reads but slow writes; B+ tree is the most common; CREATE INDEX / DROP INDEX manage them.

— End of Unit 7 —

Reminder: these notes may contain errors — verify with your textbook and teacher.