جامـــــعـة المـــــسـتـقـبـل
AL MUSTAQBAL UNIVERSITY

كلية العلوم

قســـم الامن السيبراني

Lecture: (9)

Subject: Database
Level: Second
Lecturer: Asst. Lecturer Qusai AL-Durrah

Lec 9: SQL Basics 2

Students							
StudentID	FirstName	LastName	DateOfBirth	Email	Enrollment Date	DepartmentID	GPA
1	ahmed	aljbory	22/2/1996	a@email.com	2/11/2015	4	3.2
2	zahraa	aljanabi	18/1/1996	b@email.com	3/11/2015	3	3.8
3	yusef	aljbory	25/4/1996	c@email.com	9/11/2015	1	2.9
4	zainab	alsaffar	29/9/1996	d@email.com	2/11/2015	2	2.5

UPDATE Statements: Modifying Existing Data

The UPDATE statement allows you to modify existing records in a table. This operation is essential for maintaining accurate and current data in your database as information changes over time.

Basic UPDATE Syntax

The fundamental structure of an UPDATE statement includes the table name, the SET clause specifying which columns to modify, and typically a WHERE clause to target specific rows:

```
UPDATE TableName SET Column1 = Value1, Column2 = Value2, ... WHERE
Condition;
```

Simple Update Examples

```
-- Update a student's email address

UPDATE Students SET Email = 'john.smith.new@email.com' WHERE StudentID
= 1001;

-- Update multiple columns for a single student

UPDATE Students SET DepartmentID = 3, GPA = 3.85 WHERE StudentID
= 1002;
```

The Critical Importance of the WHERE Clause

Warning: Omitting the WHERE clause will update EVERY row in the table. This is rarely the intended outcome and can lead to catastrophic data changes.

```
-- This will change EVERY student's GPA to 0.0 - almost certainly not what you want!

UPDATE Students SET GPA = 0.0;

-- Without WHERE, this affects all rows

-- Correct approach to update all students in a specific department UPDATE Students SET GPA = GPA + 0.1 WHERE DepartmentID = 2;
```

A best practice is to first run a SELECT with your WHERE condition to verify which rows will be affected before executing the UPDATE.

DELETE and TRUNCATE: Removing Data

Efficient data management involves not only adding and modifying data but also removing unnecessary or outdated records. SQL Server provides two primary mechanisms for data removal: DELETE and TRUNCATE. Understanding the differences between these commands is crucial for maintaining database performance and integrity.

The DELETE Statement

DELETE allows for precise removal of specific rows based on conditions:

```
--Basic DELETE syntax

DELETE FROM TableName WHERE Condition;

--Delete a specific student by ID

DELETE FROM Students WHERE StudentID = 1001

-- Delete multiple students matching criteria

DELETE FROM Students WHERE DepartmentID = 3 AND GPA < 2.0;
```

Critical Warning: Similar to UPDATE, omitting the WHERE clause in a DELETE statement will remove ALL rows from the table. Always verify your WHERE condition with a SELECT statement first.

DELETE operations are:

- Fully logged - Each row deletion is recorded in the transaction log
- Reversible - Can be rolled back within a transaction
- Row-by-row - Processes each row individually
- Conditional - Can target specific rows with WHERE

The TRUNCATE TABLE Statement

For situations where you need to remove ALL data from a table while preserving its structure, TRUNCATE TABLE is more efficient:

```
-- Remove all records from a table
TRUNCATE TABLE Students;
```

TRUNCATE operations are:

Minimally logged - Only page deallocations are recorded, not individual rows

Faster - Generally performs better than DELETE for removing all rows

Resource-efficient - Uses less transaction log space

Structure-preserving - Maintains the table and its columns

All-or-nothing - Cannot include a WHERE clause; removes all data

Key Differences Between DELETE and TRUNCATE

Feature	DELETE	TRUNCATE
Conditional removal	Yes (WHERE clause)	No (all rows removed)
Transaction log impact	Records each row deletion	Minimal logging
Performance	Slower for large datasets	Faster for full table removal
Identity reset	No reset of identity columns	Resets identity counters
Triggers	Activates DELETE triggers	Does not activate triggers
Rollback capability	Can be rolled back	Can be rolled back
Permission requirements	DELETE permission	Requires ALTER TABLE permission

Choosing Primary Keys in SQL Server

Primary keys are fundamental to relational database design, serving as unique identifiers for each record in a table. They enforce entity integrity and establish relationships between tables. Selecting appropriate primary keys is a critical decision that impacts database performance, maintainability, and data integrity.

Primary Key Fundamentals

A primary key must be:

- Unique - No duplicate values allowed
- Non-null - Cannot contain NULL values
- Stable - Should rarely or never change
- Minimal - Use the fewest columns necessary

Creating Primary Keys

Primary keys can be defined during table creation or added later:

```
-- Creating a table with a primary key
CREATE TABLE Students (
StudentID INT NOT NULL PRIMARY KEY,
FirstName VARCHAR(50) NOT NULL,
LastName VARCHAR(50) NOT NULL,
-- Other columns
);

-- Alternative syntax with CONSTRAINT naming CREATE TABLE Students (
StudentID INT NOT NULL,
FirstName VARCHAR(50) NOT NULL,
LastName VARCHAR(50) NOT NULL,
-- Other columns
CONSTRAINT PK_Students PRIMARY KEY (StudentID)
);

-- Adding a primary key to an existing table ALTER TABLE Students
ADD CONSTRAINT PK_Students PRIMARY KEY (StudentID);
```

Using IDENTITY for Auto-incrementing Keys

The IDENTITY property automatically generates sequential values for primary keys:

```
-- IDENTITY(seed, increment)
-- seed: starting value
-- increment: value to add for each new record CREATE TABLE Courses (
CourseID INT IDENTITY(1, 1) PRIMARY KEY,
CourseCode VARCHAR(10) UNIQUE NOT NULL,
CourseName VARCHAR(100) NOT NULL,
-- Other columns
);
```

Practice Exercises and Review

Consolidating your SQL knowledge requires hands-on practice. The following exercises combine multiple SQL operations to reinforce your understanding of database manipulation in SQL Server. These exercises progress from basic to more complex operations, allowing you to build confidence in your SQL skills.

Exercise 1: Database Setup and Basic Operations

```
-- Create a database for a small library system
CREATE DATABASE LibrarySystem;
USE LibrarySystem;

-- Create tables with appropriate relationships
CREATE TABLE Authors (
AuthorID INT IDENTITY(1,1) PRIMARY KEY, FirstName VARCHAR(50) NOT NULL, LastName VARCHAR(50) NOT NULL,
BirthYear INT
);

CREATE TABLE Books (
BookID INT IDENTITY(1,1) PRIMARY KEY, Title VARCHAR(100) NOT NULL,
AuthorID INT REFERENCES Authors(AuthorID), PublicationYear INT,
ISBN VARCHAR(13) UNIQUE,
Available BIT DEFAULT 1
);

-- Insert sample data
INSERT INTO Authors (FirstName, LastName, BirthYear) VALUES
('Jane', 'Austen', 1775),
('George', 'Orwell', 1903),
('J.K.', 'Rowling', 1965);

INSERT INTO Books (Title, AuthorID, PublicationYear, ISBN) VALUES
('Pride and Prejudice', 1, 1813, '9780141439518'),
('Nineteen Eighty-Four', 2, 1949, '9780451524935'),
('Harry Potter and the Philosopher''s Stone', 3, 1997, '9780747532699');

-- Practice querying data
SELECT * from Books, Authors
```