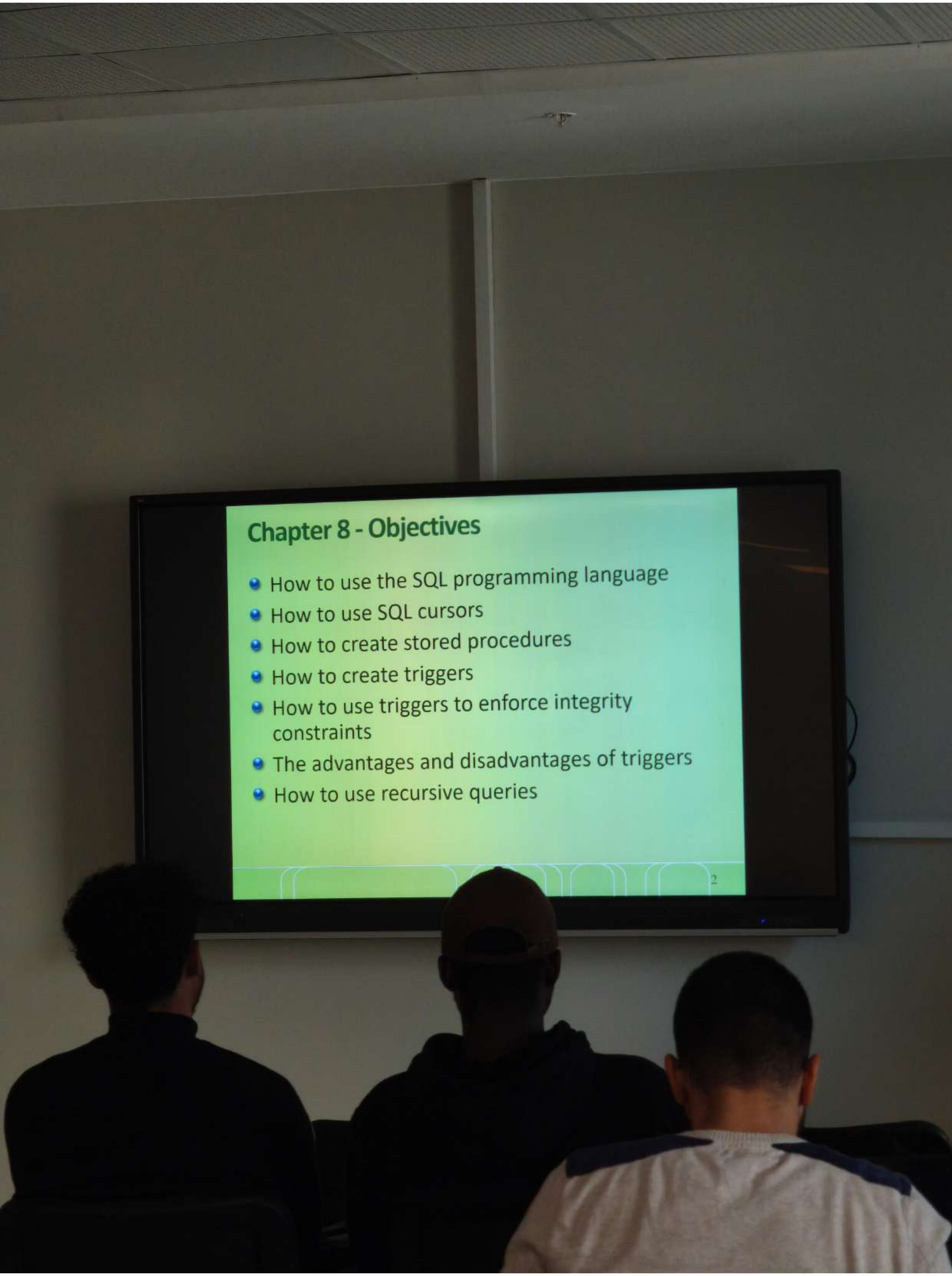
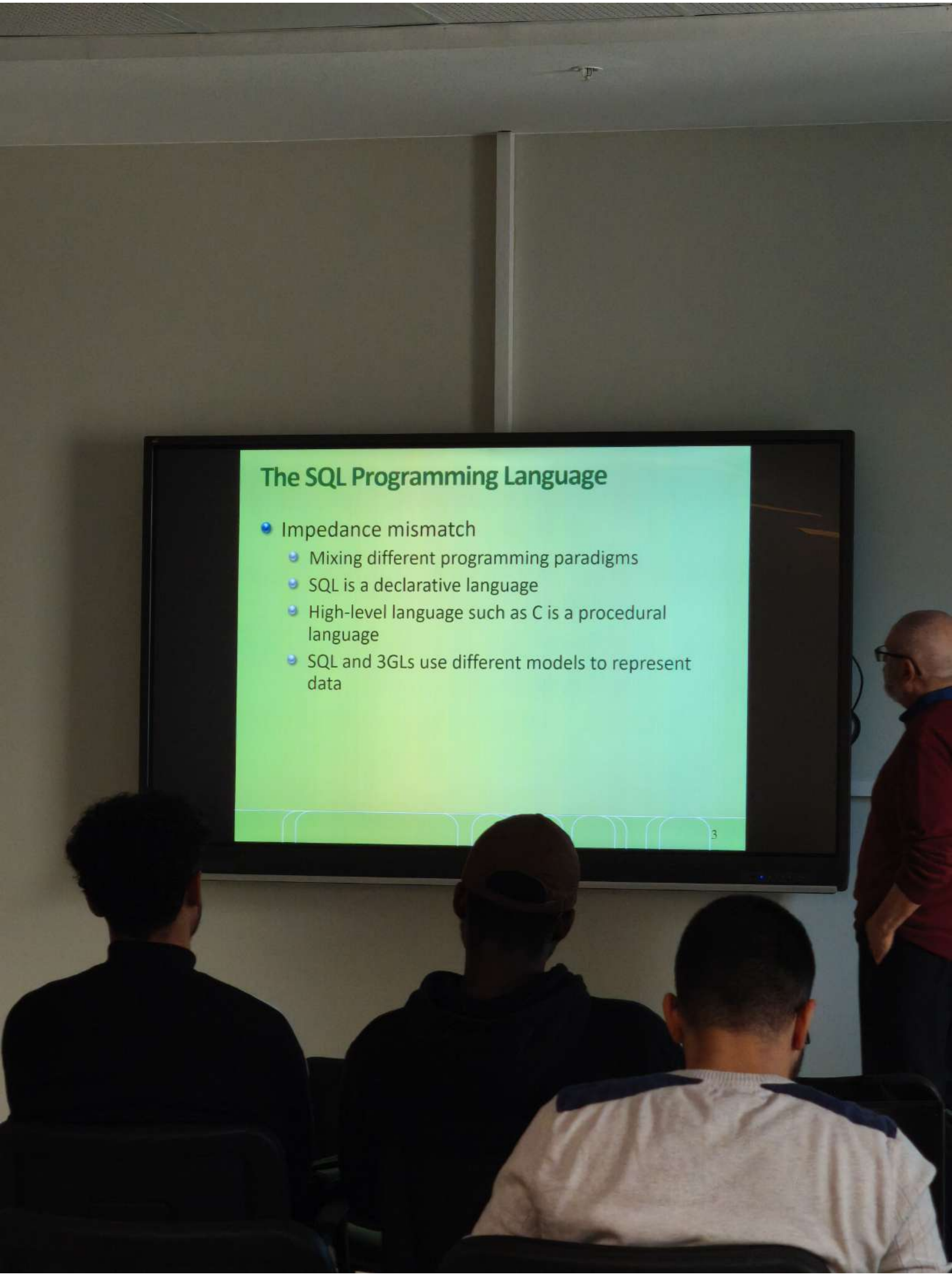


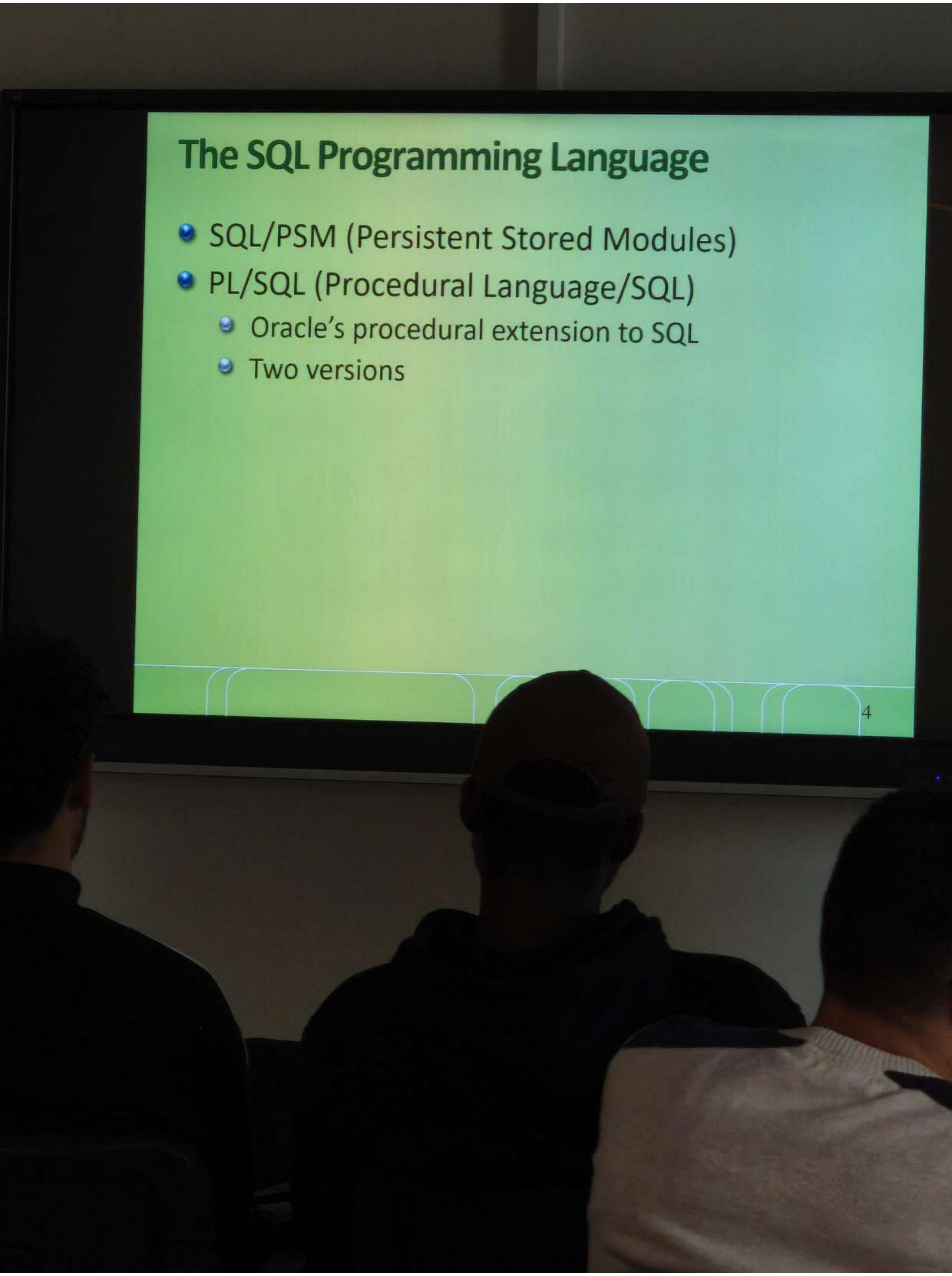
Chapter 8 - Objectives

- How to use the SQL programming language
- How to use SQL cursors
- How to create stored procedures
- How to create triggers
- How to use triggers to enforce integrity constraints
- The advantages and disadvantages of triggers
- How to use recursive queries



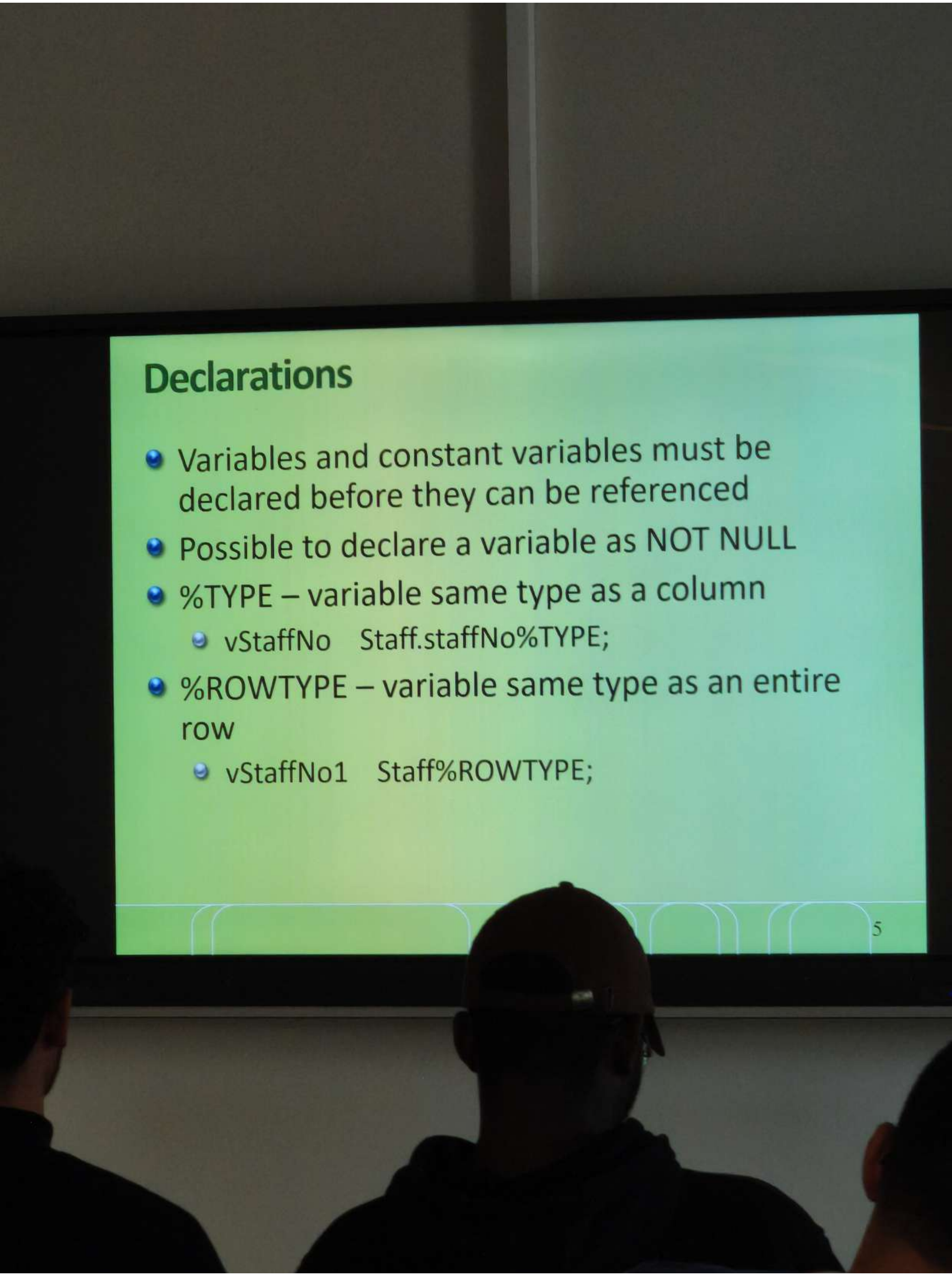
The SQL Programming Language

- Impedance mismatch
  - Mixing different programming paradigms
  - SQL is a declarative language
  - High-level language such as C is a procedural language
  - SQL and 3GLs use different models to represent data



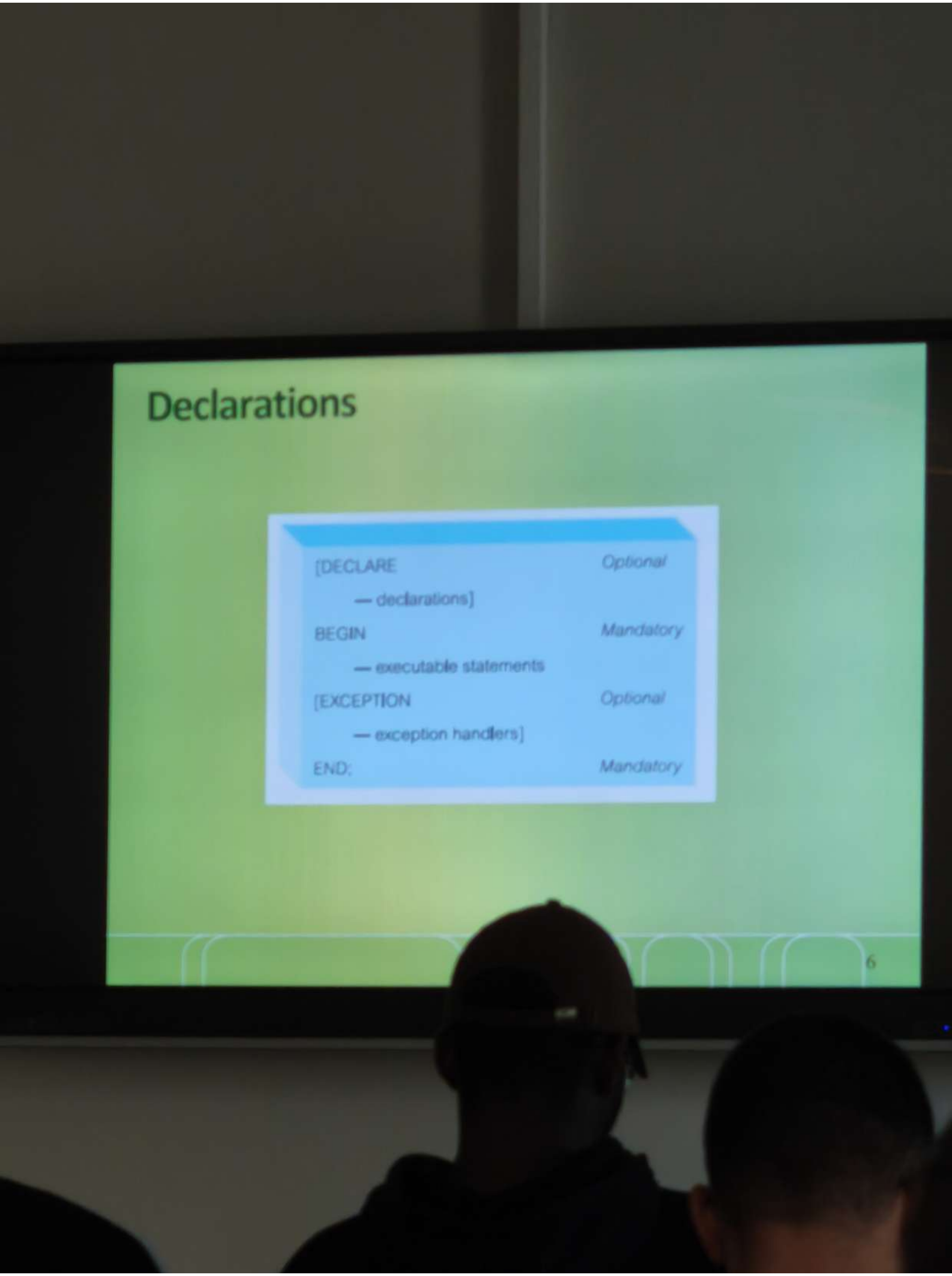
The SQL Programming Language

- SQL/PSM (Persistent Stored Modules)
- PL/SQL (Procedural Language/SQL)
  - Oracle's procedural extension to SQL
  - Two versions



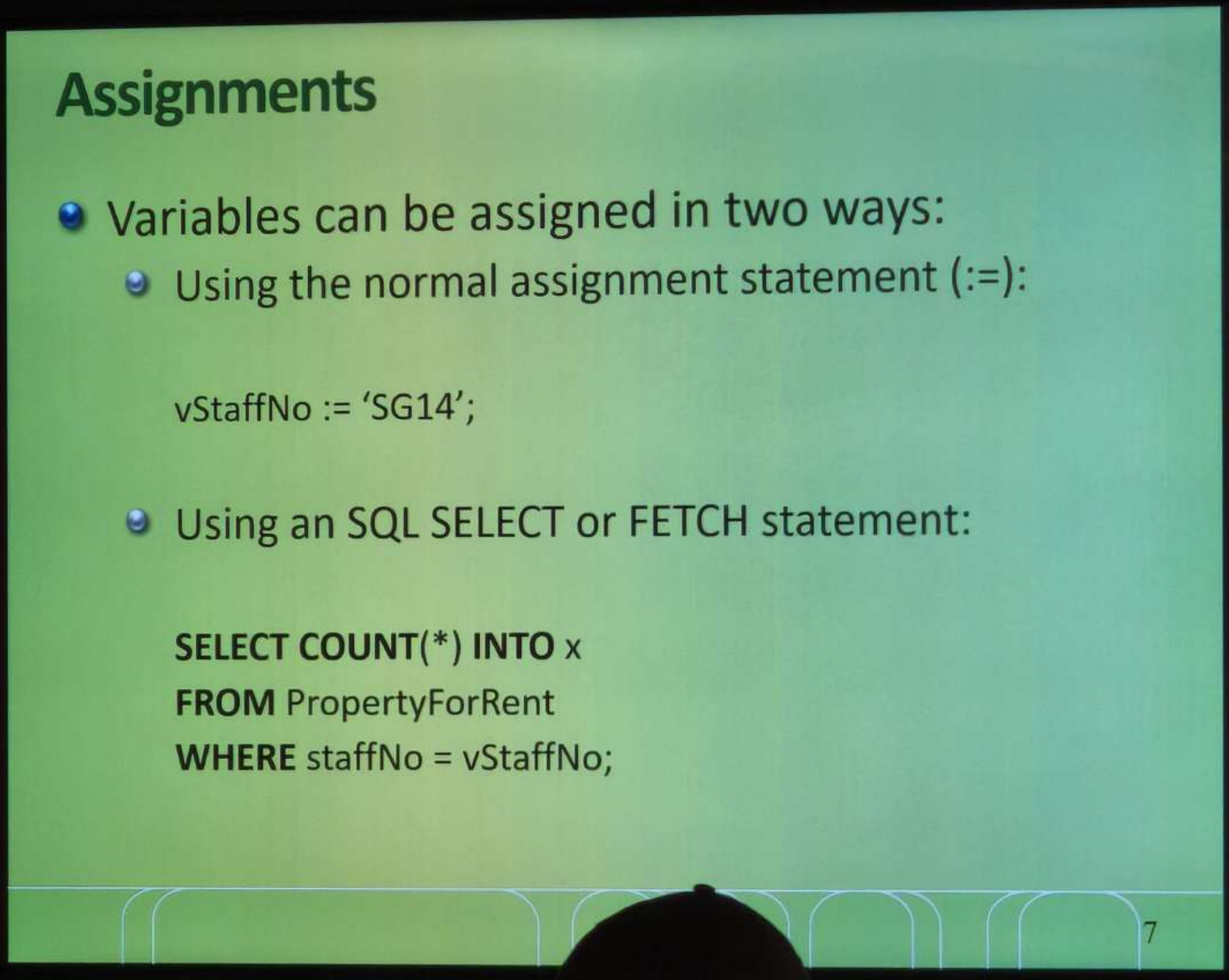
Declarations

- Variables and constant variables must be declared before they can be referenced
- Possible to declare a variable as NOT NULL
- %TYPE – variable same type as a column
  - vStaffNo   Staff.staffNo%TYPE;
- %ROWTYPE – variable same type as an entire row
  - vStaffNo1   Staff%ROWTYPE;



Declarations

```
[DECLARE                              Optional
      — declarations]
BEGIN                                 Mandatory
      — executable statements
[EXCEPTION                            Optional
      — exception handlers]
END;                                  Mandatory
```



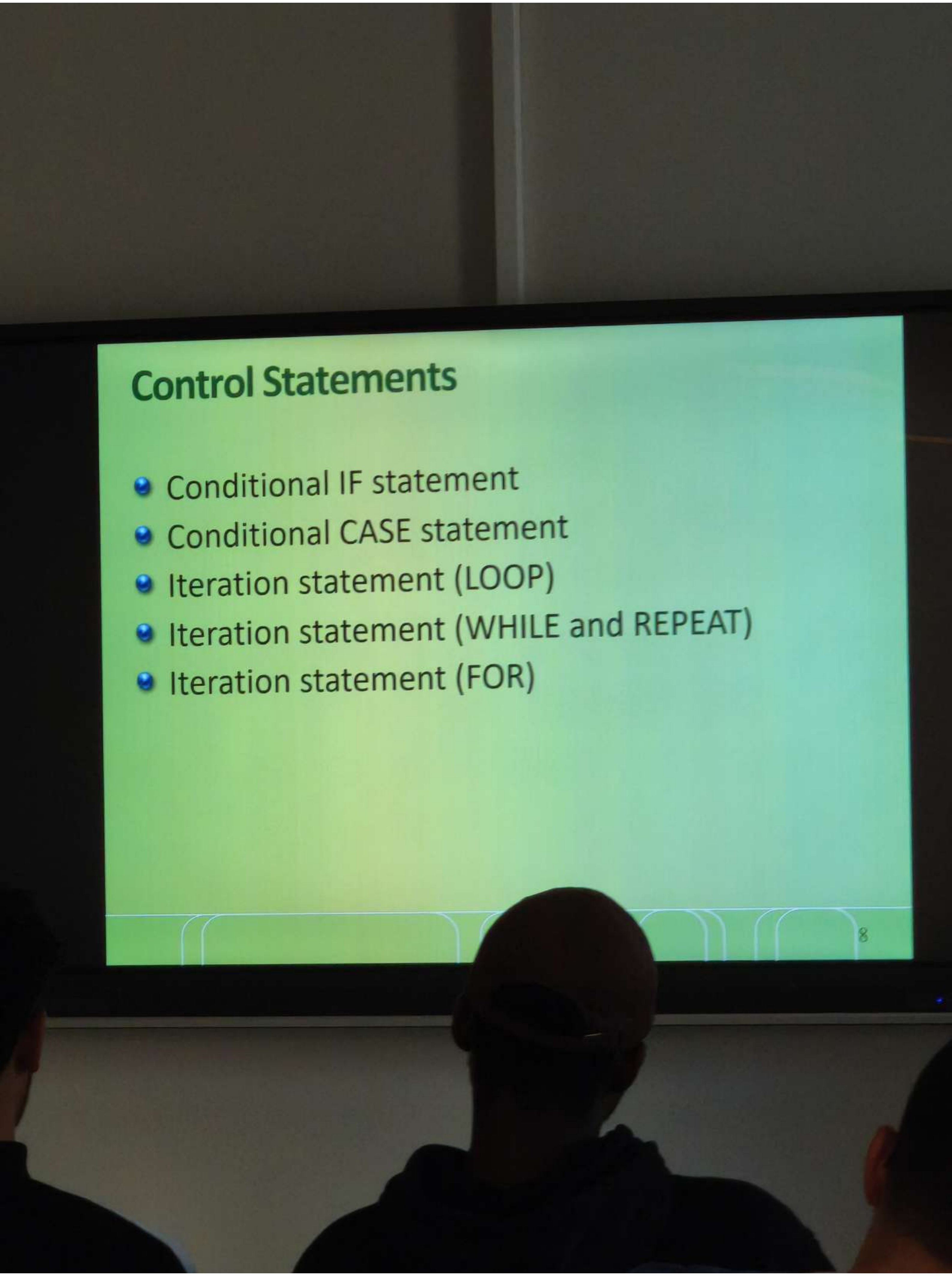
Assignments

- Variables can be assigned in two ways:
  - Using the normal assignment statement (:=):

```
vStaffNo := 'SG14';
```

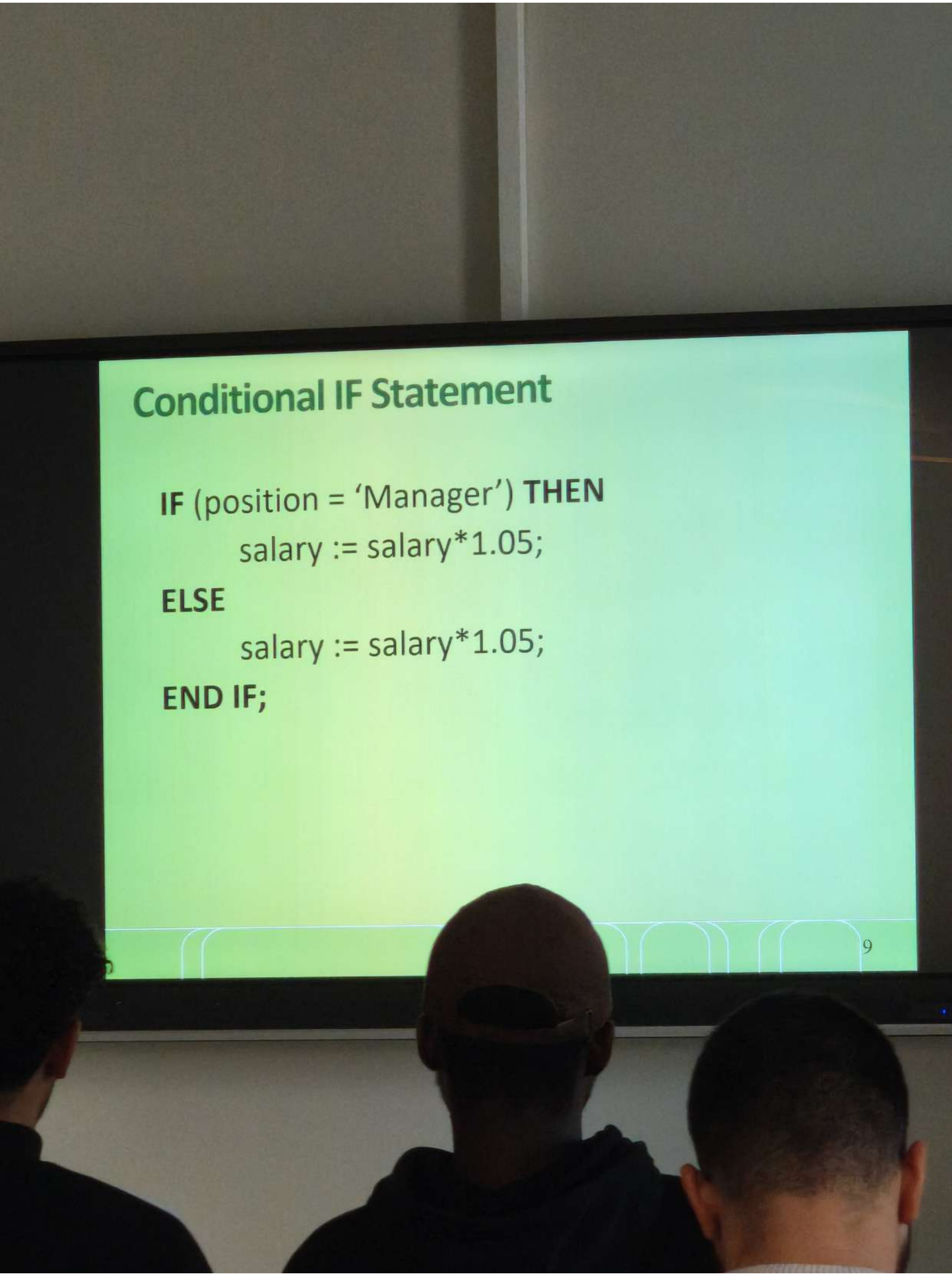
  - Using an SQL SELECT or FETCH statement:

```
SELECT COUNT(*) INTO x
FROM PropertyForRent
WHERE staffNo = vStaffNo;
```



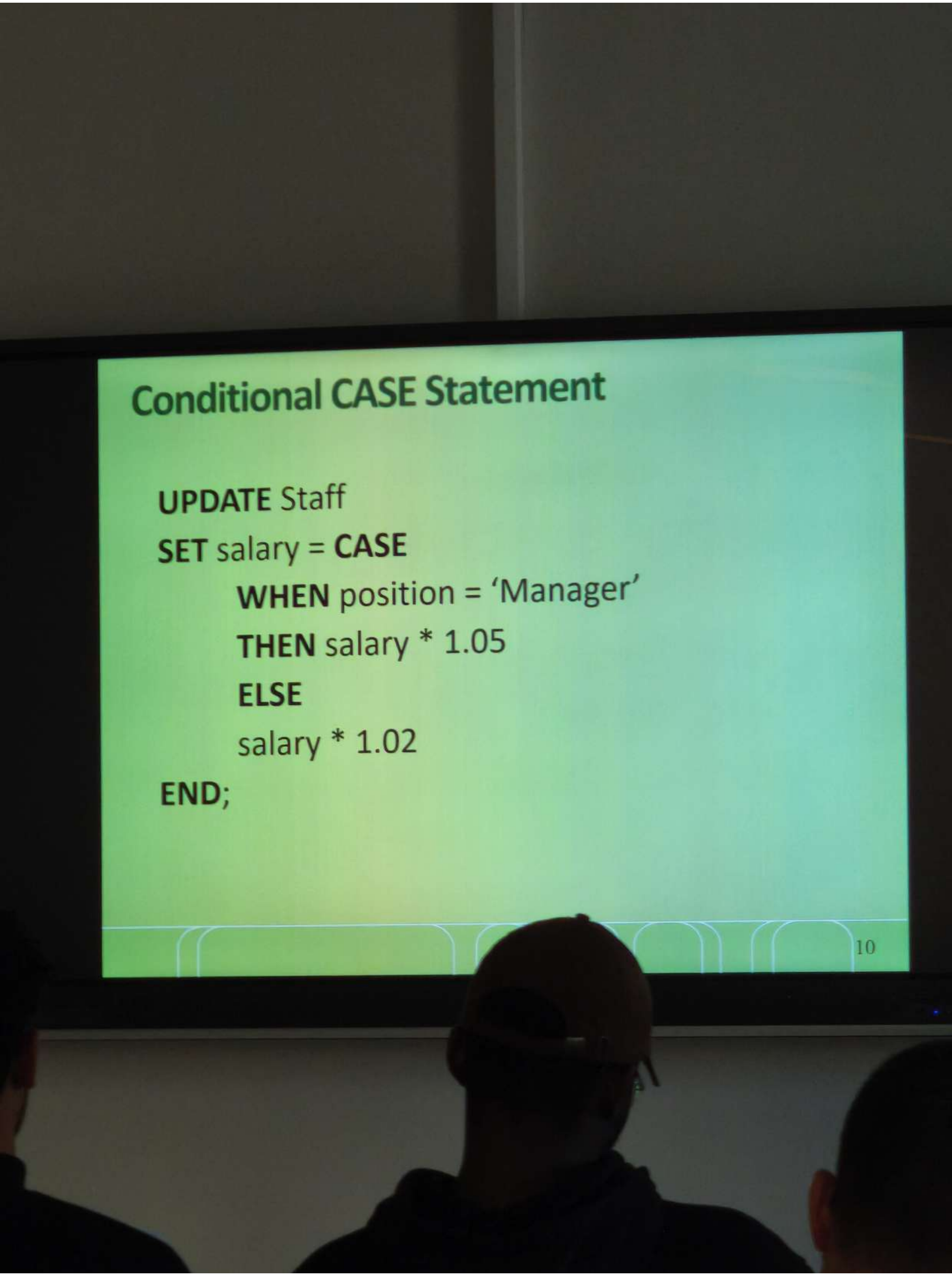
Control Statements

- Conditional IF statement
- Conditional CASE statement
- Iteration statement (LOOP)
- Iteration statement (WHILE and REPEAT)
- Iteration statement (FOR)



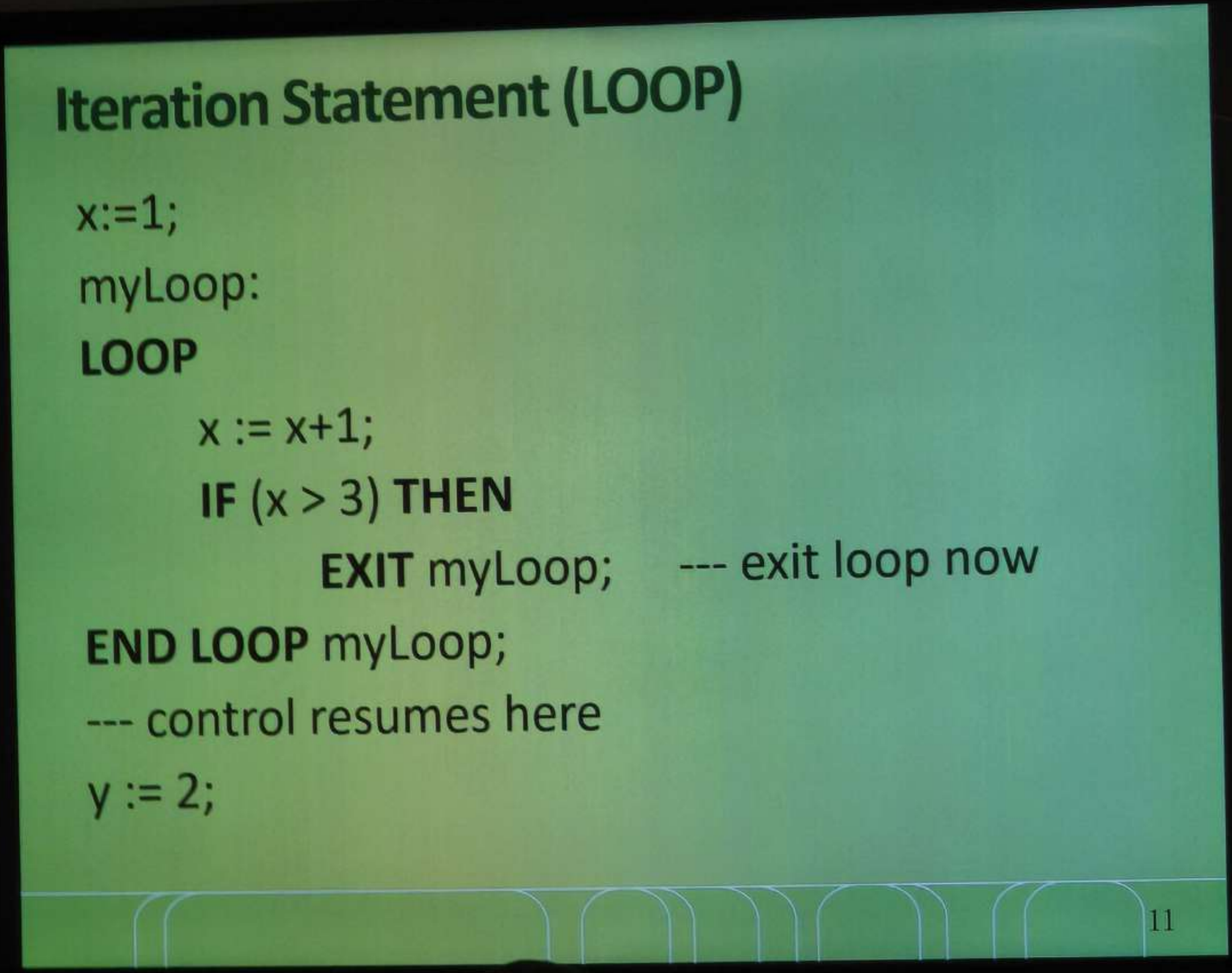
Conditional IF Statement

```
IF (position = ‘Manager’) THEN
      salary := salary*1.05;
ELSE
      salary := salary*1.05;
END IF;
```



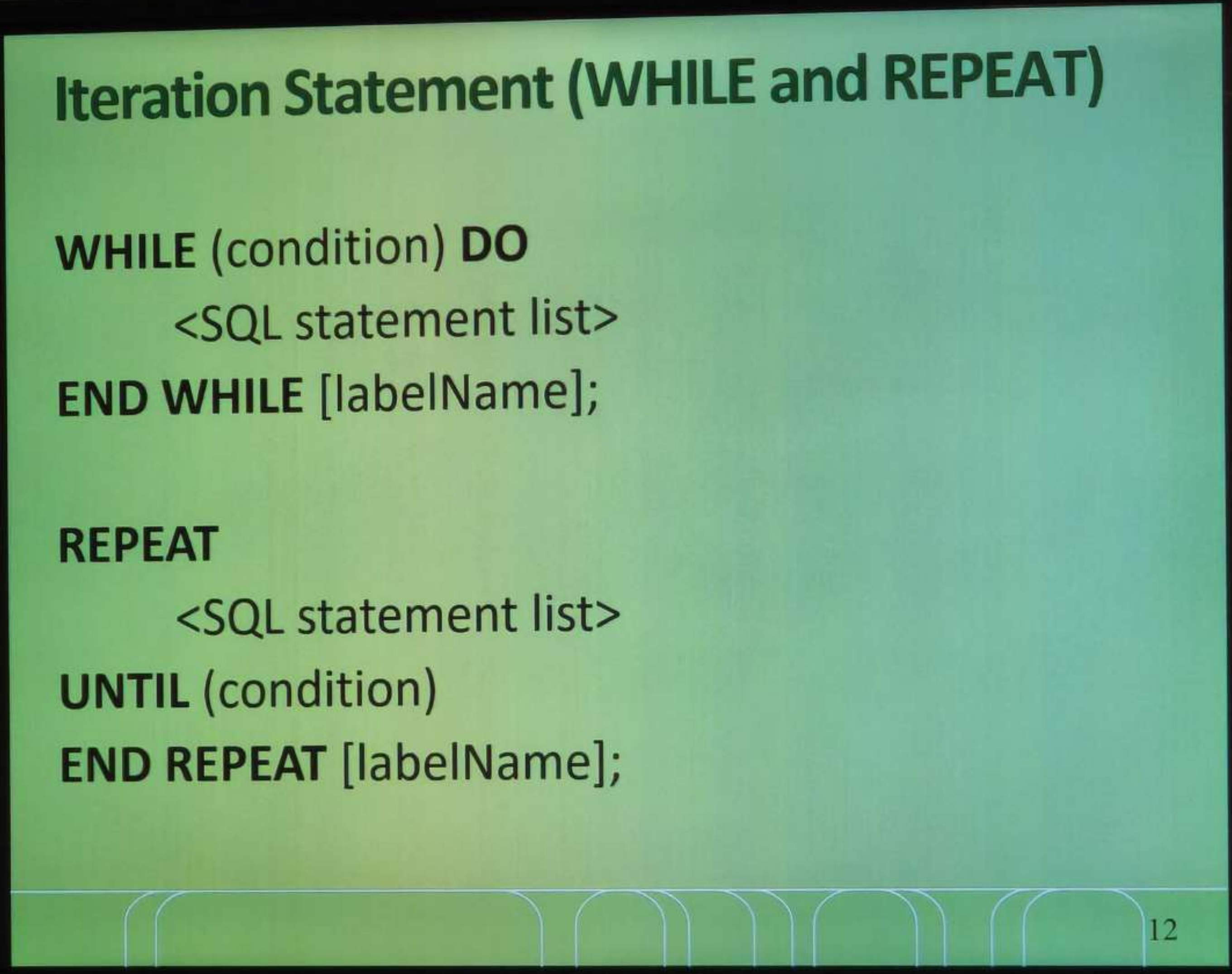
Conditional CASE Statement

```
UPDATE Staff
SET salary = CASE
        WHEN position = 'Manager'
        THEN salary * 1.05
        ELSE
        salary * 1.02
END;
```



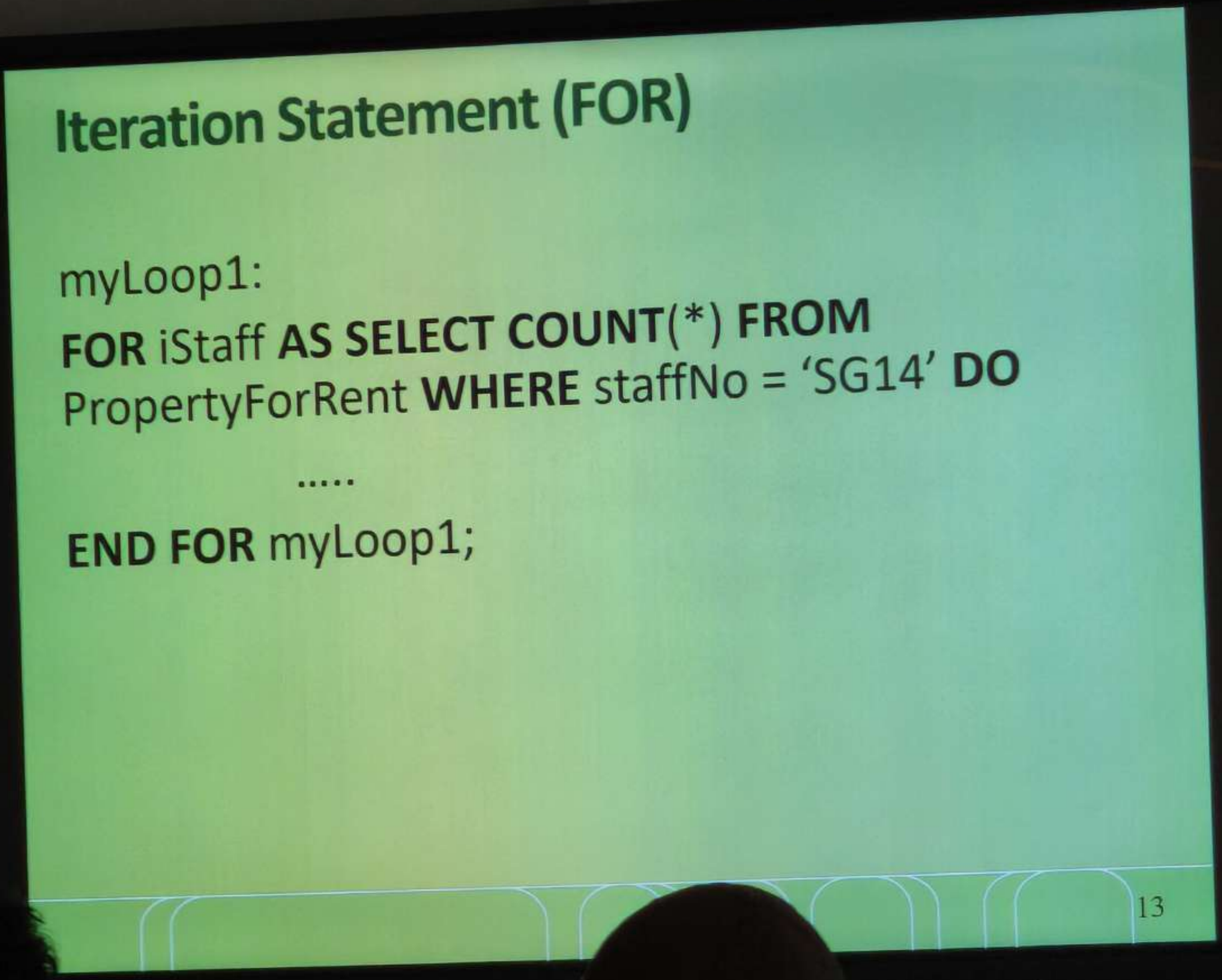
Iteration Statement (LOOP)

```
x:=1;
myLoop:
LOOP
        x := x+1;
        IF (x > 3) THEN
                EXIT myLoop;     --- exit loop now
END LOOP myLoop;
--- control resumes here
y := 2;
```



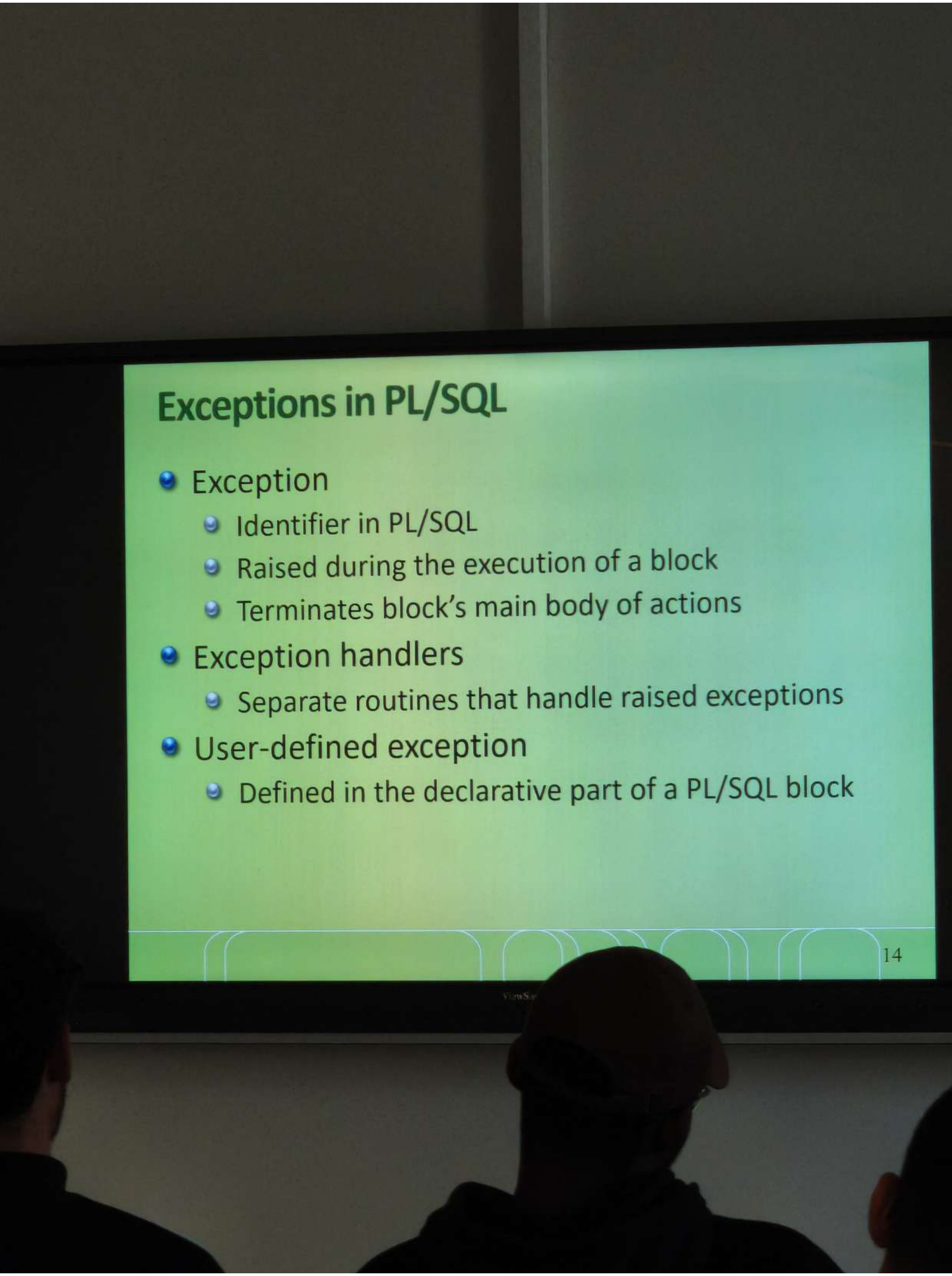
Iteration Statement (WHILE and REPEAT)

```
WHILE (condition) DO
      <SQL statement list>
END WHILE [labelName];

REPEAT
      <SQL statement list>
UNTIL (condition)
END REPEAT [labelName];
```



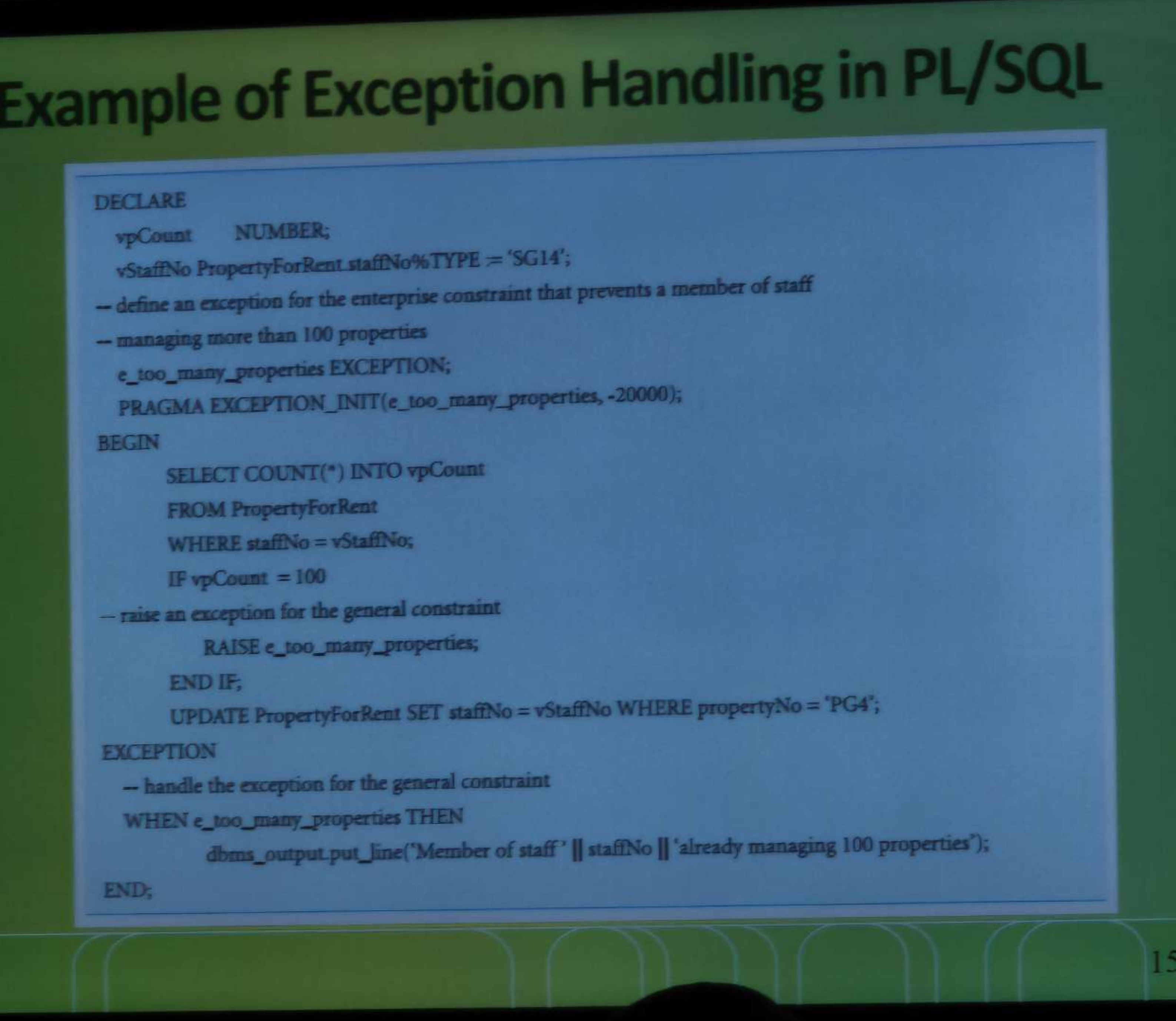
Iteration Statement (FOR)

```
myLoop1:
FOR iStaff AS SELECT COUNT(*) FROM
PropertyForRent WHERE staffNo = 'SG14' DO
        .....
END FOR myLoop1;
```



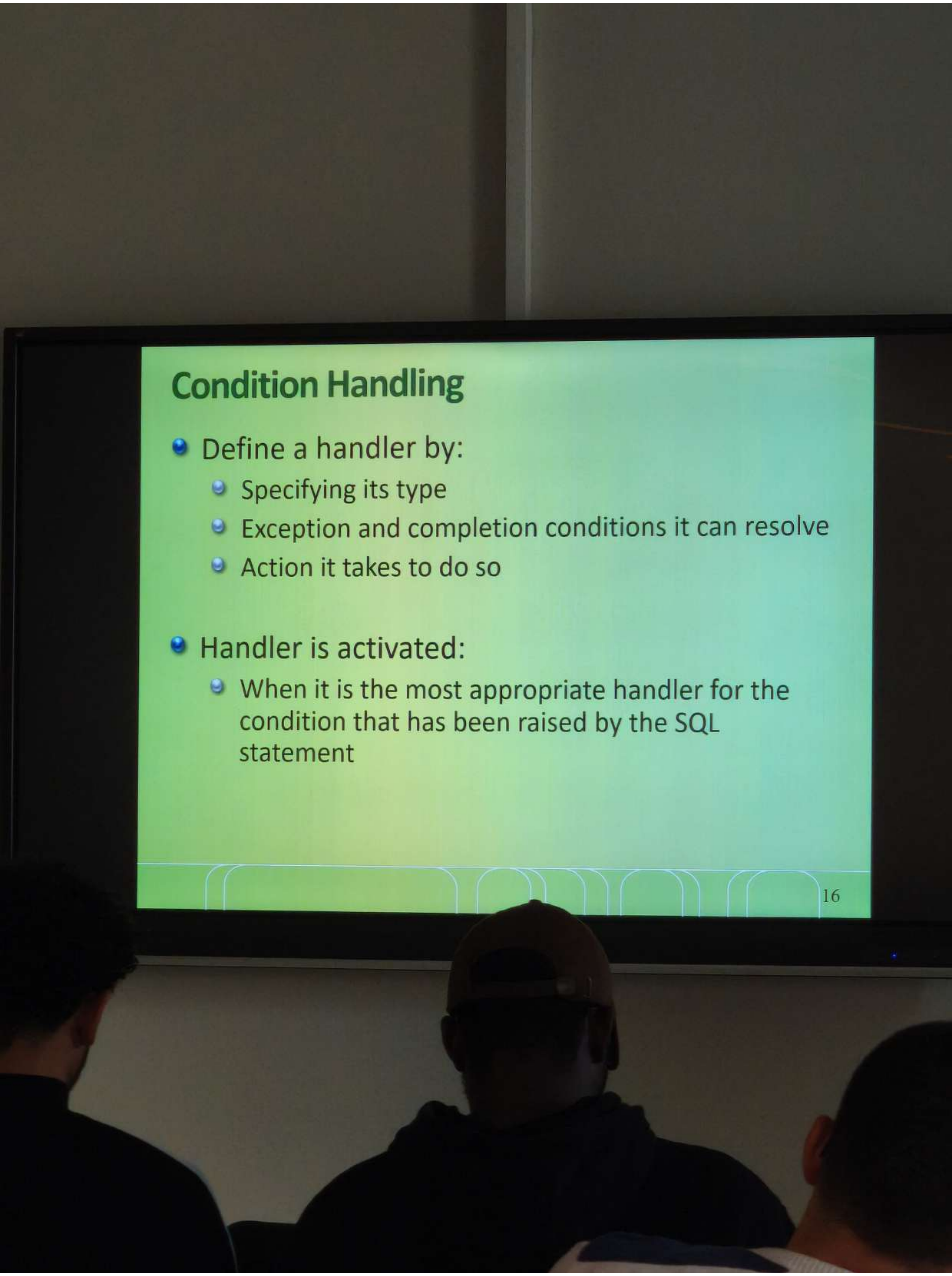
Exceptions in PL/SQL

- Exception
  - Identifier in PL/SQL
  - Raised during the execution of a block
  - Terminates block’s main body of actions
- Exception handlers
  - Separate routines that handle raised exceptions
- User-defined exception
  - Defined in the declarative part of a PL/SQL block



Example of Exception Handling in PL/SQL

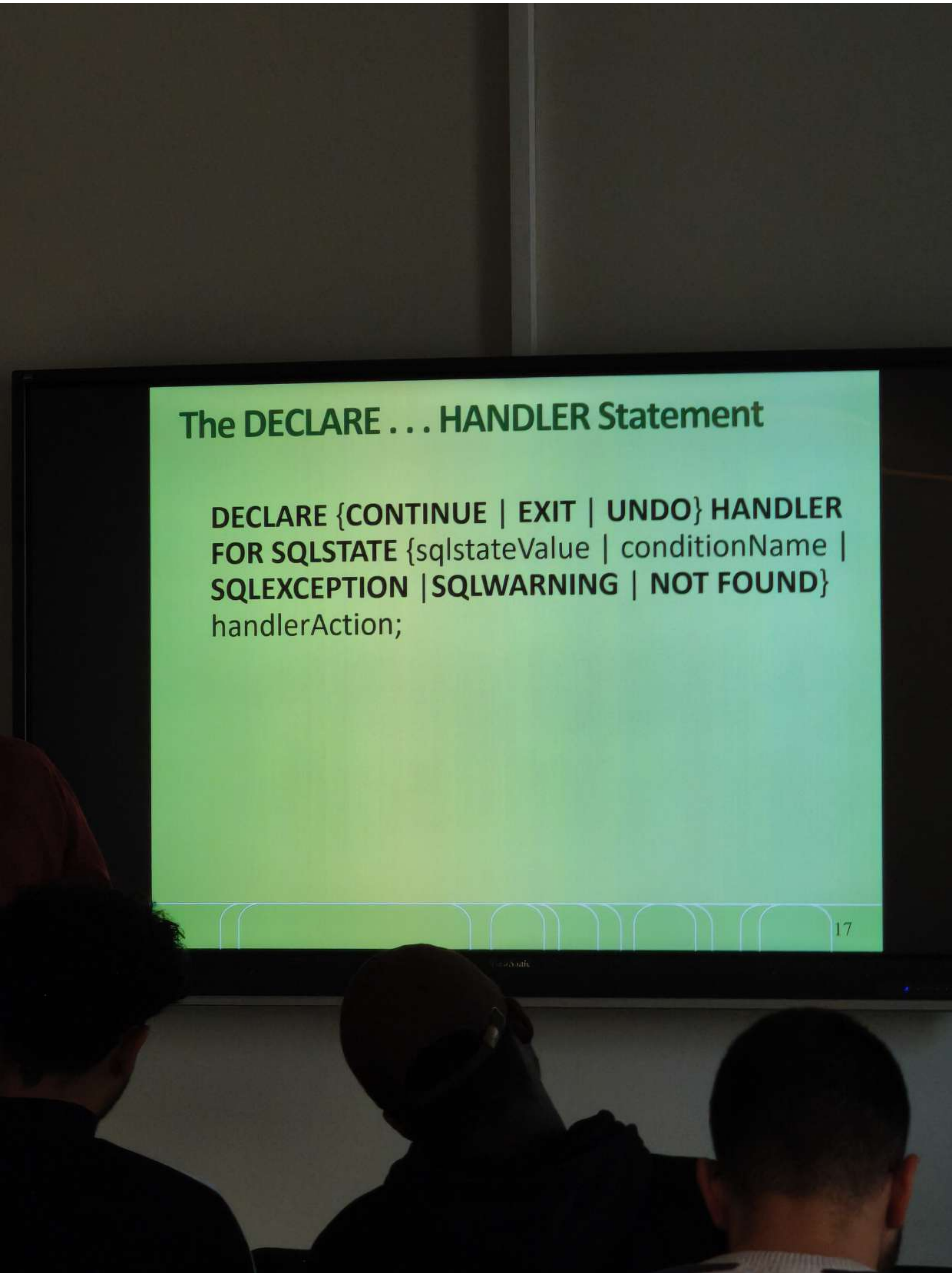
```
DECLARE
  vpCount      NUMBER;
  vStaffNo PropertyForRent.staffNo%TYPE := 'SG14';
-- define an exception for the enterprise constraint that prevents a member of staff
-- managing more than 100 properties
  e_too_many_properties EXCEPTION;
  PRAGMA EXCEPTION_INIT(e_too_many_properties, -20000);
BEGIN
       SELECT COUNT(*) INTO vpCount
       FROM PropertyForRent
       WHERE staffNo = vStaffNo;
       IF vpCount  = 100
-- raise an exception for the general constraint
            RAISE e_too_many_properties;
       END IF;
       UPDATE PropertyForRent SET staffNo = vStaffNo WHERE propertyNo = 'PG4';
EXCEPTION
  -- handle the exception for the general constraint
  WHEN e_too_many_properties THEN
            dbms_output.put_line('Member of staff' || staffNo || 'already managing 100 properties');
END;
```



Condition Handling

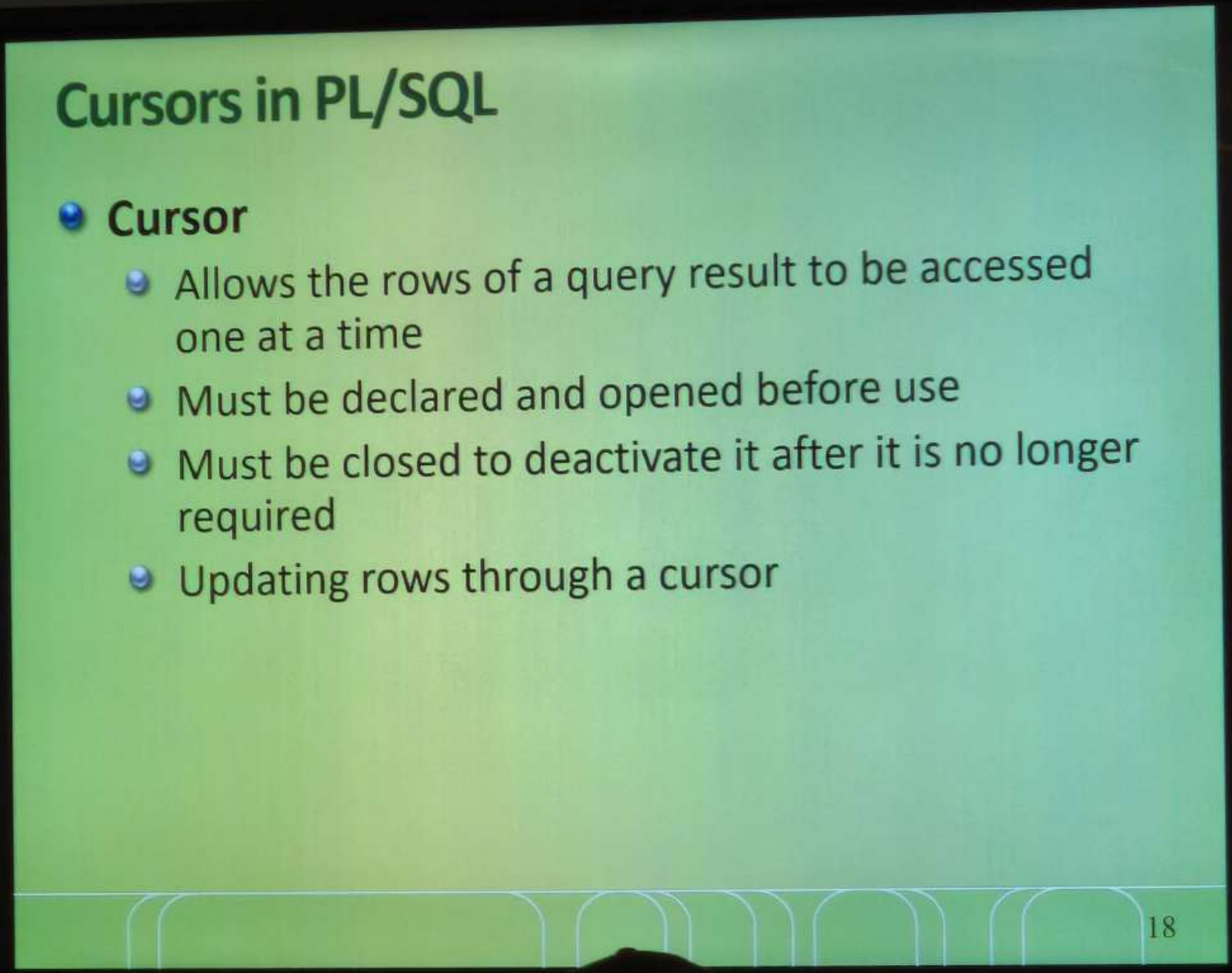
- Define a handler by:
  - Specifying its type
  - Exception and completion conditions it can resolve
  - Action it takes to do so

- Handler is activated:
  - When it is the most appropriate handler for the condition that has been raised by the SQL statement



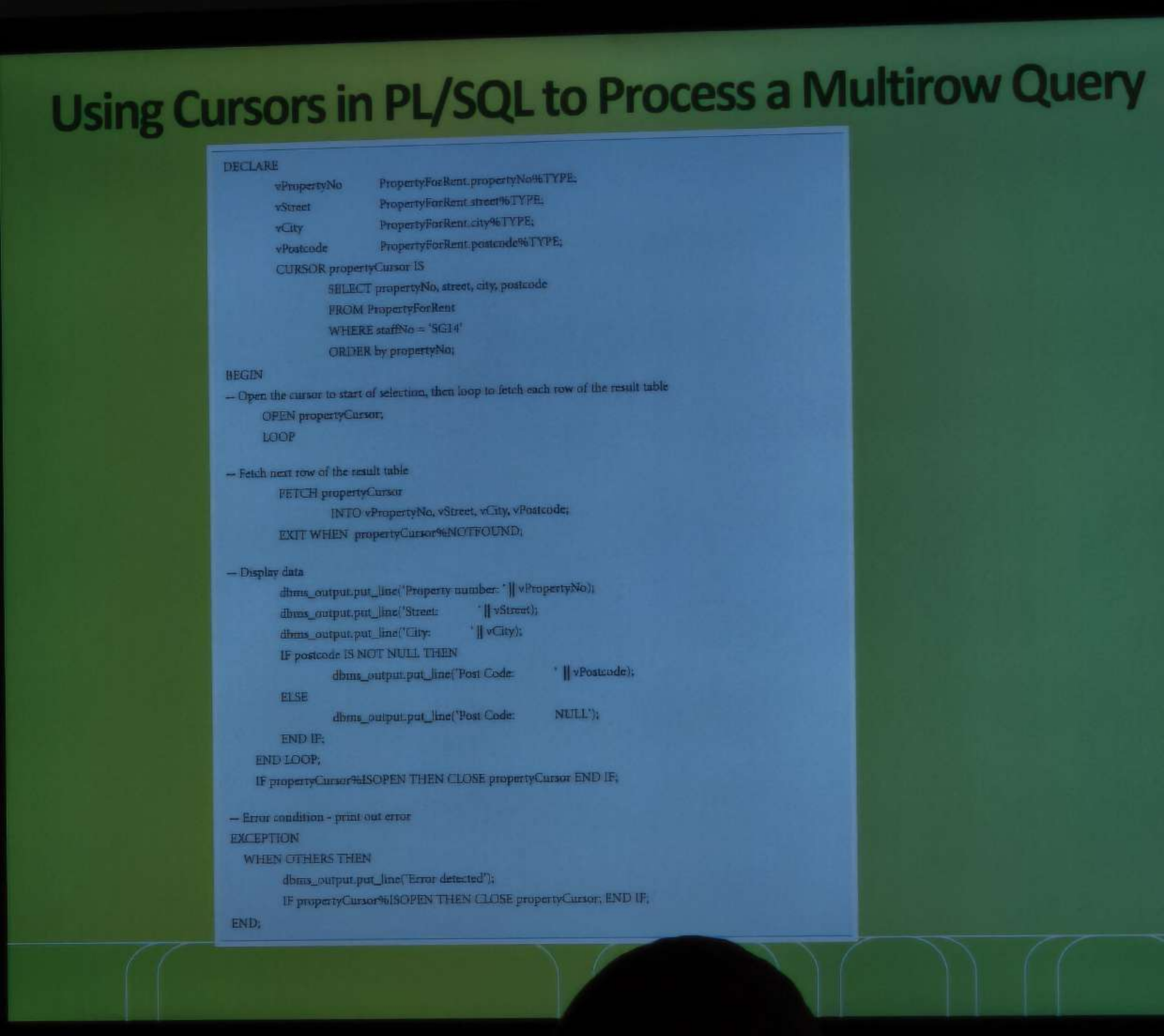
The DECLARE . . . HANDLER Statement

DECLARE {**CONTINUE** | **EXIT** | **UNDO**} **HANDLER FOR SQLSTATE** {sqlstateValue | conditionName | **SQLEXCEPTION** |**SQLWARNING** | **NOT FOUND**} handlerAction;



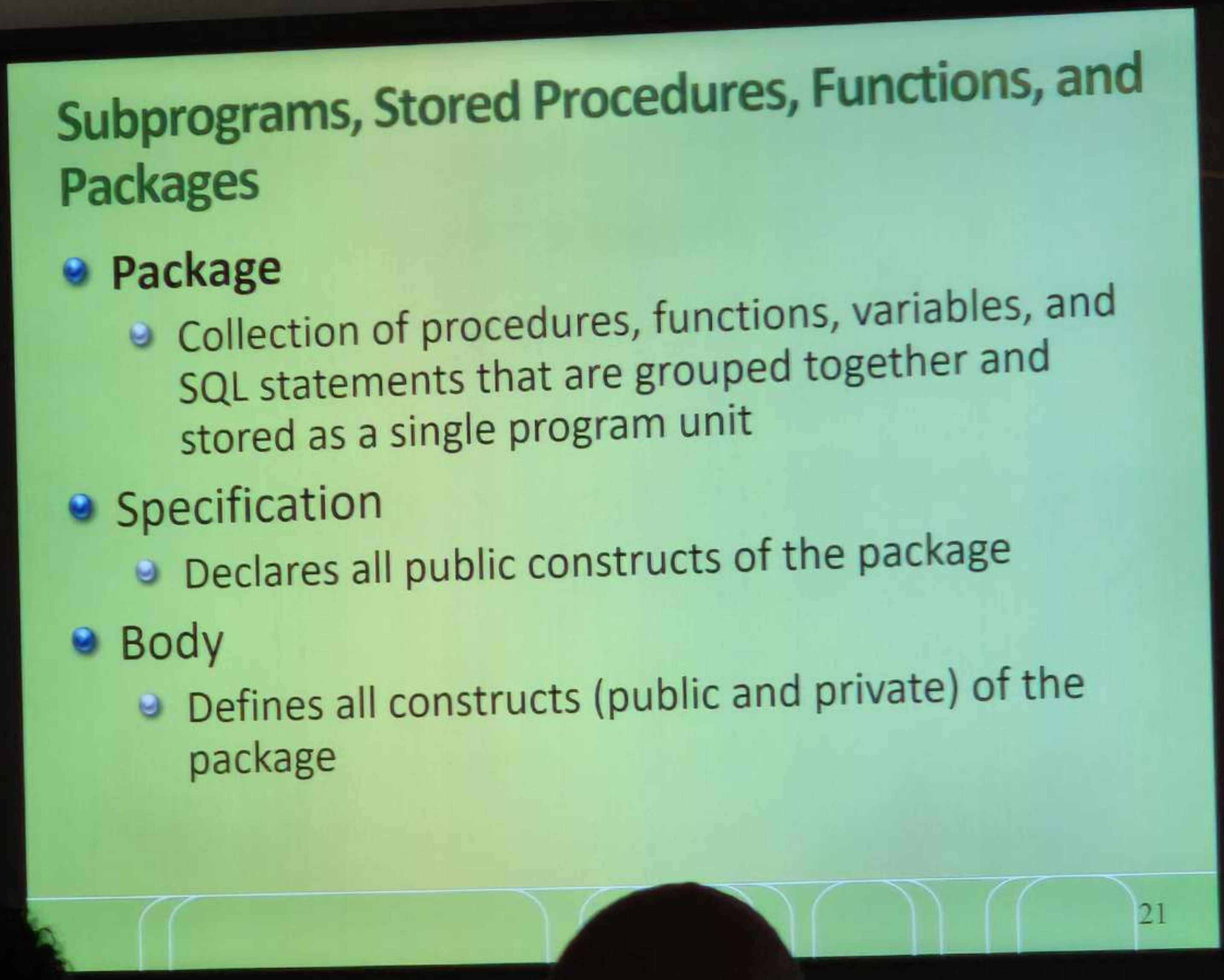
Cursors in PL/SQL

- **Cursor**
  - Allows the rows of a query result to be accessed one at a time
  - Must be declared and opened before use
  - Must be closed to deactivate it after it is no longer required
  - Updating rows through a cursor



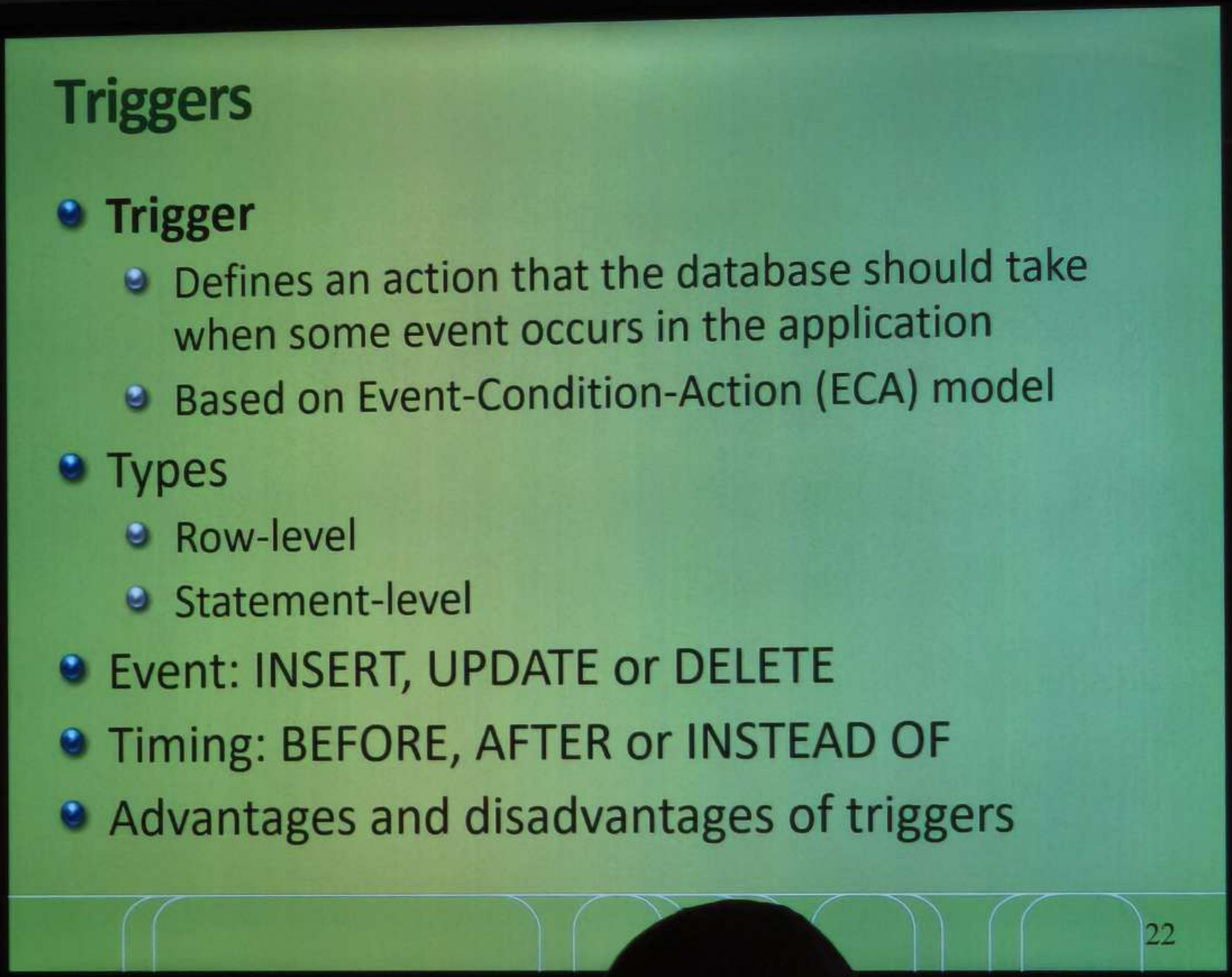
Using Cursors in PL/SQL to Process a Multirow Query

```
DECLARE
        vPropertyNo         PropertyForRent.propertyNo%TYPE;
        vStreet             PropertyForRent.street%TYPE;
        vCity               PropertyForRent.city%TYPE;
        vPostcode           PropertyForRent.postcode%TYPE;
        CURSOR propertyCursor IS
                SELECT propertyNo, street, city, postcode
                FROM PropertyForRent
                WHERE staffNo = 'SG14'
                ORDER by propertyNo;
BEGIN
-- Open the cursor to start of selection, then loop to fetch each row of the result table
      OPEN propertyCursor;
      LOOP

-- Fetch next row of the result table
        FETCH propertyCursor
                INTO vPropertyNo, vStreet, vCity, vPostcode;
        EXIT WHEN  propertyCursor%NOTFOUND;

-- Display data
        dbms_output.put_line('Property number: ' || vPropertyNo);
        dbms_output.put_line('Street:          ' || vStreet);
        dbms_output.put_line('City:            ' || vCity);
        IF postcode IS NOT NULL THEN
                dbms_output.put_line('Post Code:          ' || vPostcode);
        ELSE
                dbms_output.put_line('Post Code:          NULL');
        END IF;
    END LOOP;
    IF propertyCursor%ISOPEN THEN CLOSE propertyCursor END IF;

-- Error condition - print out error
EXCEPTION
  WHEN OTHERS THEN
        dbms_output.put_line('Error detected');
        IF propertyCursor%ISOPEN THEN CLOSE propertyCursor; END IF;
END;
```



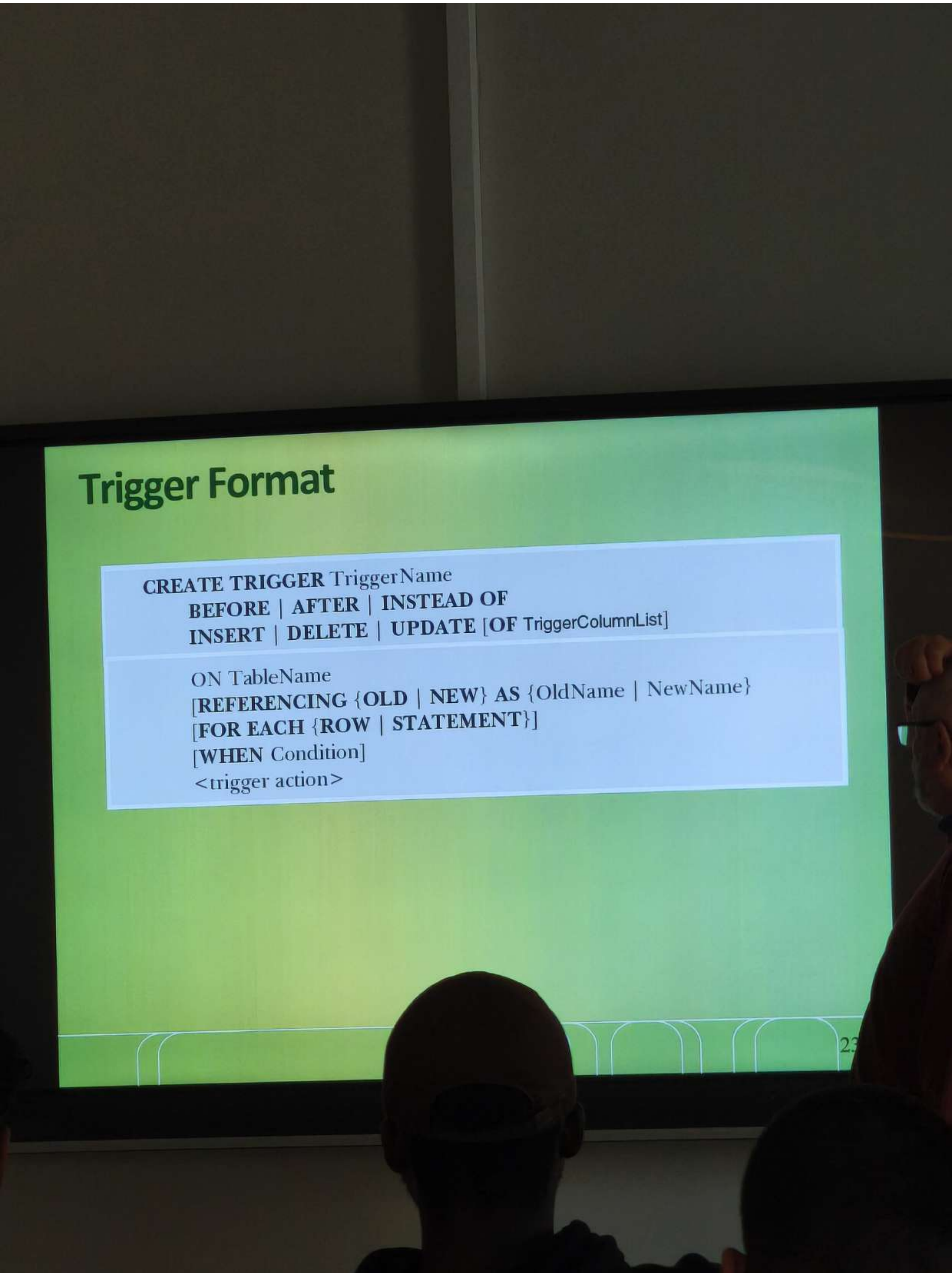
Subprograms, Stored Procedures, Functions, and Packages

- **Package**
  - Collection of procedures, functions, variables, and SQL statements that are grouped together and stored as a single program unit
- Specification
  - Declares all public constructs of the package
- Body
  - Defines all constructs (public and private) of the package



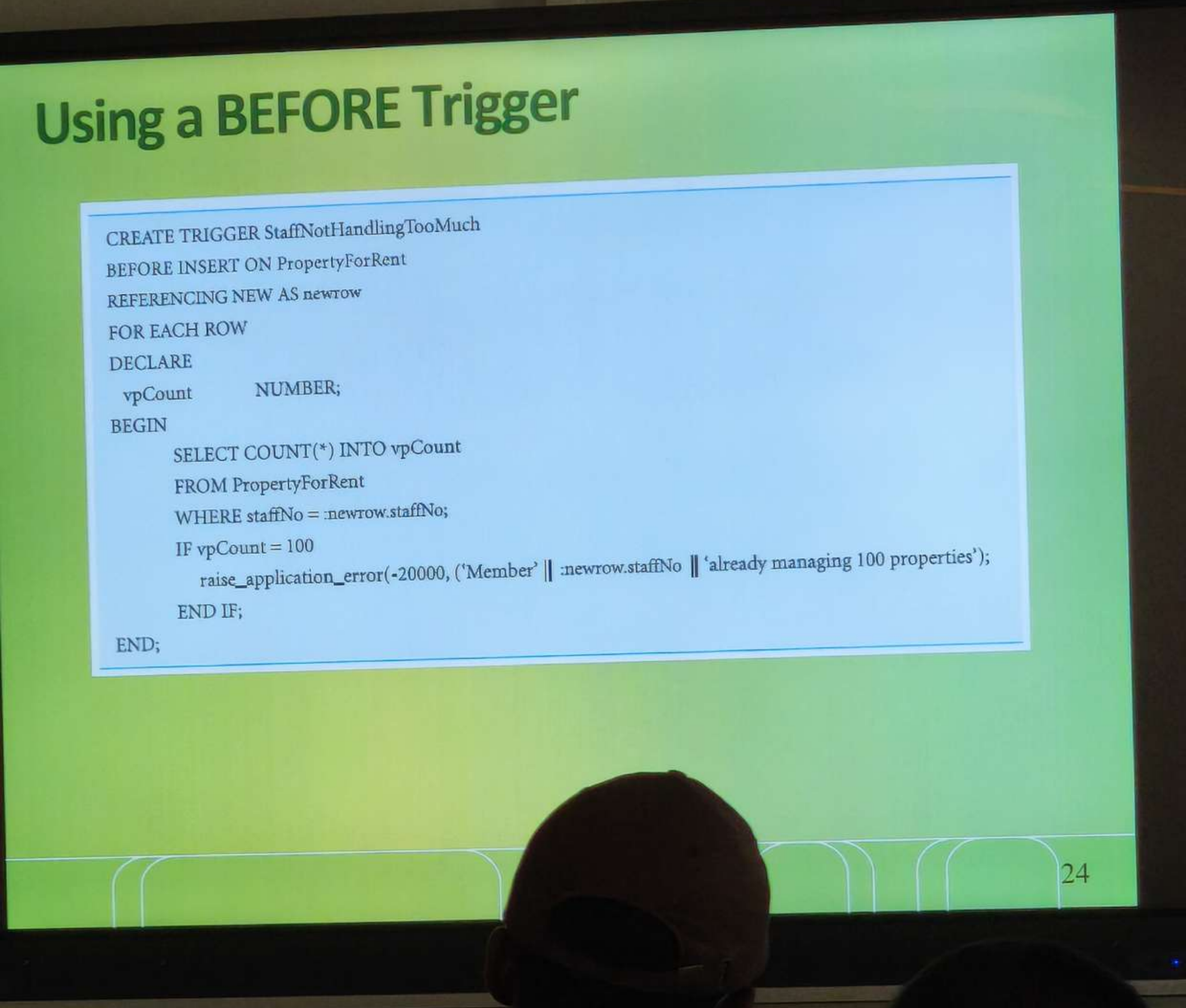
Triggers

- **Trigger**
  - Defines an action that the database should take when some event occurs in the application
  - Based on Event-Condition-Action (ECA) model
- Types
  - Row-level
  - Statement-level
- Event: INSERT, UPDATE or DELETE
- Timing: BEFORE, AFTER or INSTEAD OF
- Advantages and disadvantages of triggers



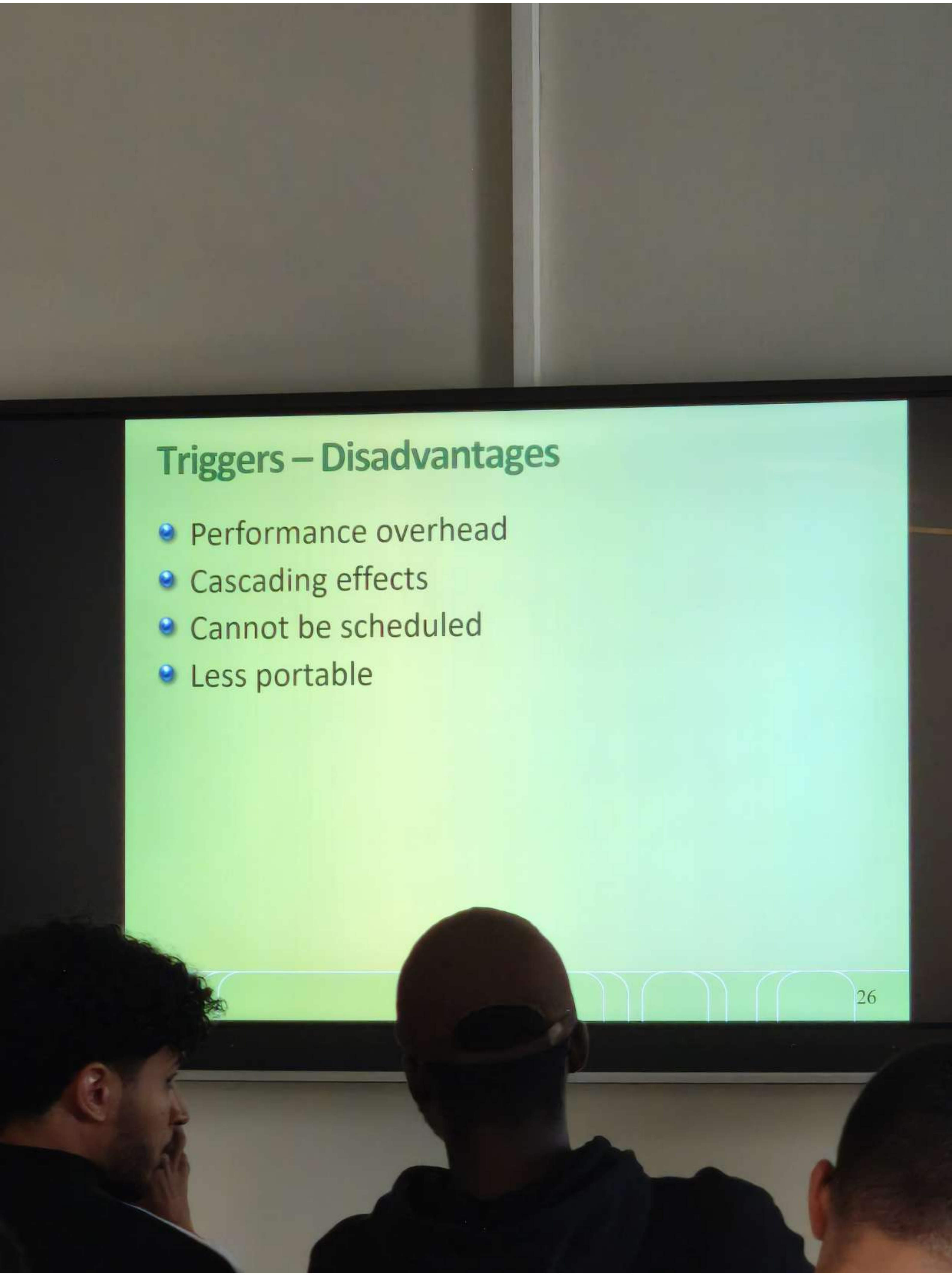
Trigger Format

```
CREATE TRIGGER TriggerName
      BEFORE | AFTER | INSTEAD OF
      INSERT | DELETE | UPDATE [OF TriggerColumnList]
      ON TableName
      [REFERENCING {OLD | NEW} AS {OldName | NewName}
      [FOR EACH {ROW | STATEMENT}]
      [WHEN Condition]
      <trigger action>
```



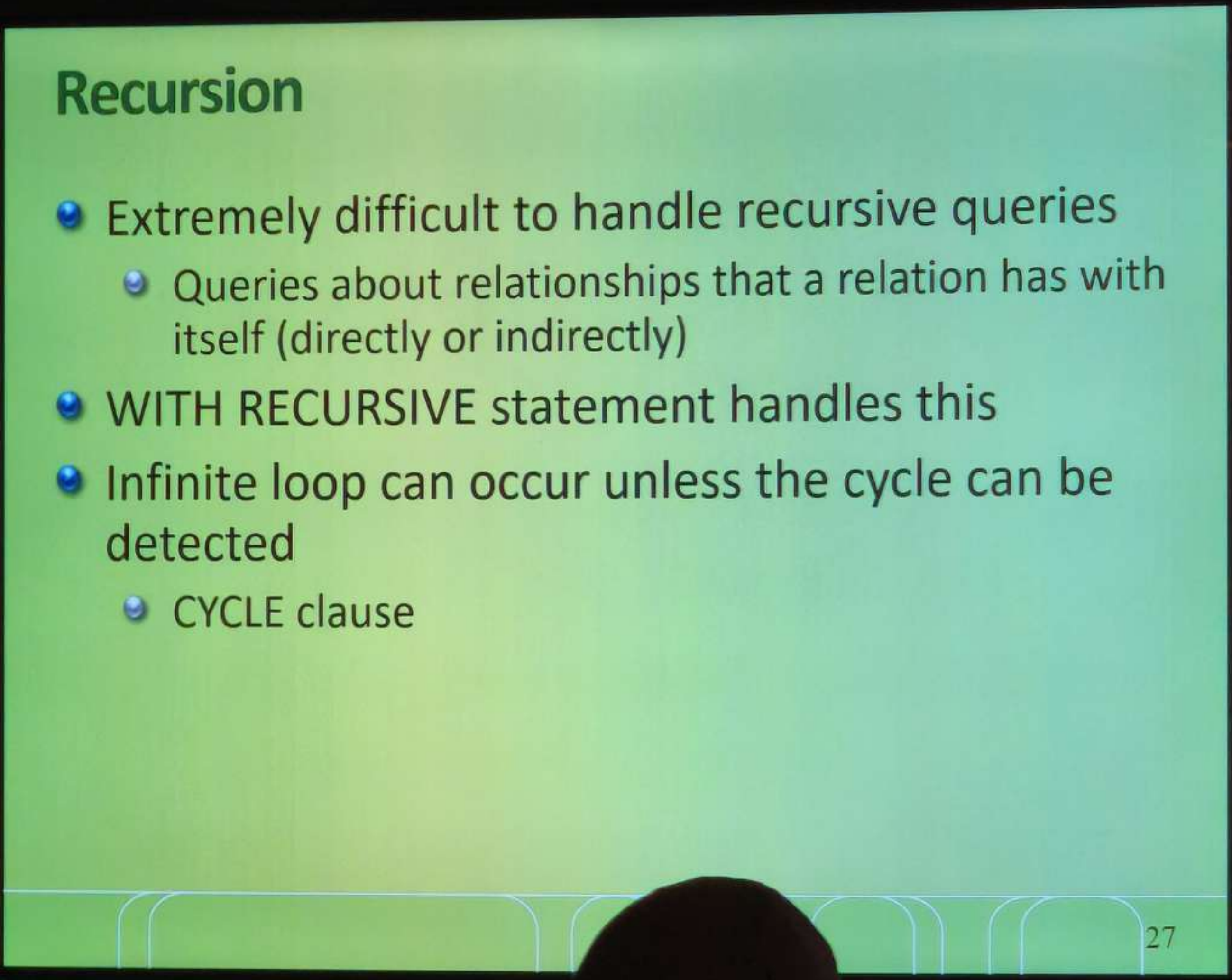
Using a BEFORE Trigger

```
CREATE TRIGGER StaffNotHandlingTooMuch
BEFORE INSERT ON PropertyForRent
REFERENCING NEW AS newrow
FOR EACH ROW
DECLARE
  vpCount        NUMBER;
BEGIN
        SELECT COUNT(*) INTO vpCount
        FROM PropertyForRent
        WHERE staffNo = :newrow.staffNo;
        IF vpCount = 100
           raise_application_error(-20000, ('Member' || :newrow.staffNo || 'already managing 100 properties');
        END IF;
END;
```



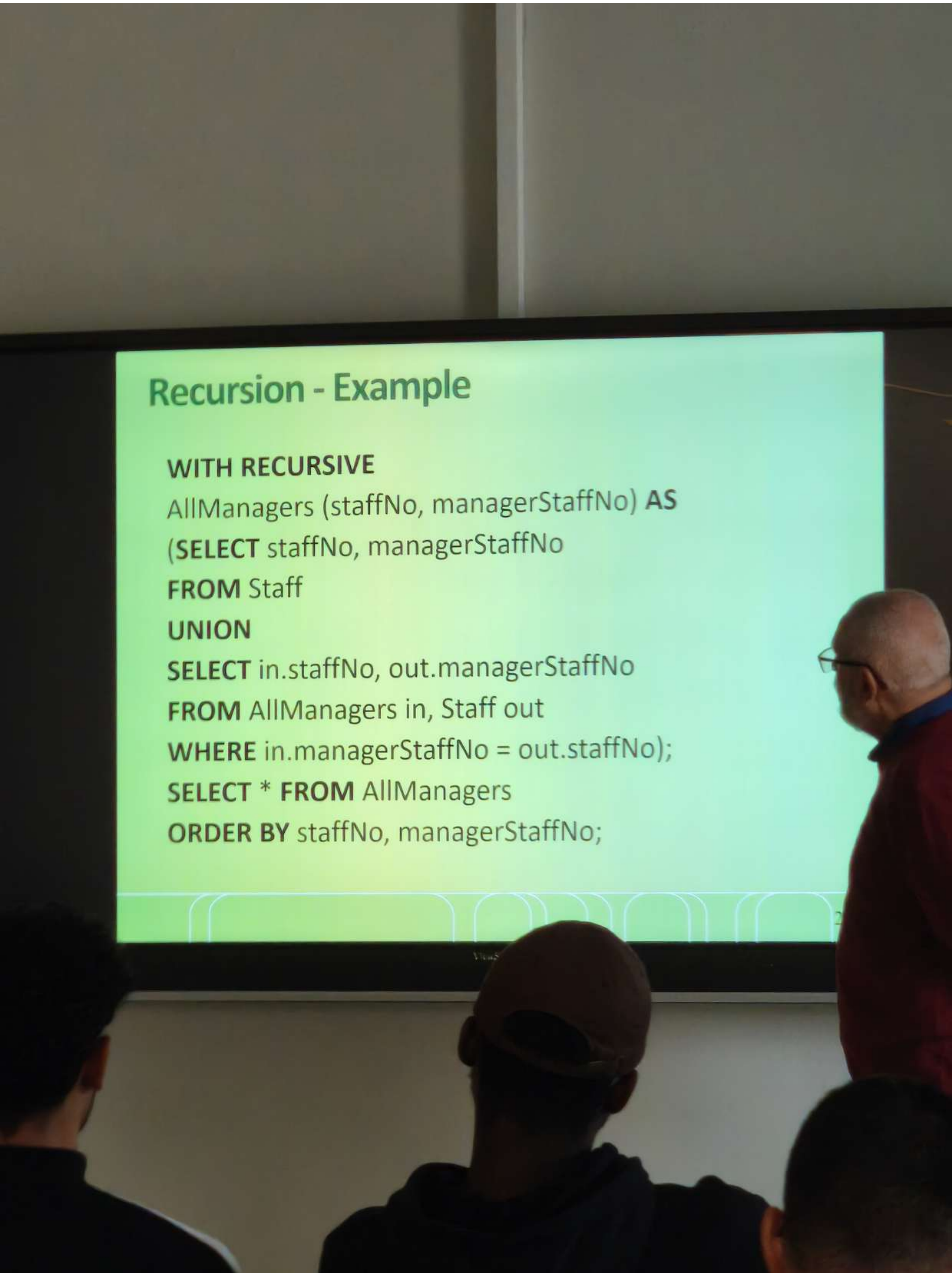
Triggers – Disadvantages

- Performance overhead
- Cascading effects
- Cannot be scheduled
- Less portable



Recursion

- Extremely difficult to handle recursive queries
  - Queries about relationships that a relation has with itself (directly or indirectly)
- WITH RECURSIVE statement handles this
- Infinite loop can occur unless the cycle can be detected
  - CYCLE clause



Recursion - Example

```
WITH RECURSIVE
AllManagers (staffNo, managerStaffNo) AS
(SELECT staffNo, managerStaffNo
FROM Staff
UNION
SELECT in.staffNo, out.managerStaffNo
FROM AllManagers in, Staff out
WHERE in.managerStaffNo = out.staffNo);
SELECT * FROM AllManagers
ORDER BY staffNo, managerStaffNo;
```