

# COMP 330: Imperative SQL 2

Chris Jermaine and Kia Teymourian  
Rice University

# More Interesting Cursor Example

Write a stored procedure giving the tallest height in a region.

```
CREATE PROCEDURE getTallestPeak  
@whichRegion VARCHAR (8000),  
@result VARCHAR (8000) OUTPUT  
AS BEGIN  
...  
END
```

# More Interesting Cursor Example

Body 1: Declare control vars as well as the cursor

```
DECLARE @peakName VARCHAR (8000);  
DECLARE @bestName VARCHAR (8000);  
DECLARE @peakHeight INT;  
DECLARE @bestHeight INT;  
SET @bestHeight = -1;  
  
DECLARE myRes CURSOR FOR  
SELECT name, elev FROM peak WHERE region = @whichRegion;
```

# More Interesting Cursor Example

Body 2: Open cursor and loop to find the tallest peak

```
OPEN myRes;  
FETCH NEXT FROM myRes INTO @peakName,  
@peakHeight;  
  
WHILE (@@FETCH_STATUS = 0) BEGIN  
    IF @peakHeight > @bestHeight BEGIN  
        SET @bestHeight = @peakHeight;  
        SET @bestName = @peakName;  
    END  
    FETCH NEXT FROM myRes INTO @peakName,  
@peakHeight;  
END
```

# Some Notes

## @@FETCH\_STATUS

- ▷ 0 means good
- ▷ -1 means fail or beyond result set
- ▷ -2 means fetched row missing

## FETCH

- ▷ FETCH NEXT
- ▷ FETCH FIRST
- ▷ FETCH LAST
- ▷ FETCH ABSOLUTE  $n$
- ▷ FETCH RELATIVE  $n$

# More Interesting Cursor Example

Body 3: return the result

```
CLOSE myRes;  
DEALLOCATE myRes;  
  
SET @result = @bestName;  
END;
```

To call:

```
DECLARE @myResult VARCHAR (8000);  
EXECUTE getTallestPeak  
    @whichRegion = 'Corocoran_to_Whitney',  
    @result = @myResult output;  
PRINT @myResult;  
GO
```

# Important: Don't EVER Write Such Code!

I wrote code that looped to find the tallest

- ▷ Terrible idea!
- ▷ TOP k would be shorter, easier, faster
- ▷ Rule: use AS MUCH declarative code as possible
- ▷ Only use loops, etc. when you MUST
- ▷ Sometimes 3+ orders of magnitude speed diff

# TSQL Functions

```
CREATE FUNCTION foo (@myArg INTEGER)  
RETURNS INTEGER AS BEGIN  
...  
RETURN (@someValue);  
...  
END;
```

▷ Why useful?

▷ Can call from within SQL statement!

```
SELECT *  
FROM SOME_TABLE s  
WHERE foo (s.SOME_ATT) = 12;
```



# TSQL Functions

Can even have table-valued functions

```
SELECT *  
FROM tableValuedFunction (234) t  
WHERE t.SOME_ATT = 12;
```

# Triggers

Stored procedures that fire in response to some event

- ▶ Ex: trigger that catches updates to peak table, prints error message and does not process

```
CREATE TRIGGER checkHeight ON peak FOR UPDATE AS BEGIN  
...  
END
```

- ▶ FOR/AFTER: run only once triggering action succeeds
- ▶ INSTEAD OF: don't run triggering action; run this instead
- ▶ Can have UPDATE, INSERT, DELETE

# Triggers

Body 1: Exit check and declarations

```
IF @@ROWCOUNT = 0 BEGIN
    RETURN;
END
```

```
DECLARE myRes CURSOR FOR
SELECT d.name, d.elev, i.elev FROM inserted i, deleted d
WHERE i.name = d.name AND i.elev <> d.elev;
DECLARE @peakName VARCHAR (8000);
DECLARE @oldHeight INT;
DECLARE @newHeight INT;
```

- ▷ @@ROWCOUNT: number of tuples affected
- ▷ deleted: table containing old versions of records
- ▷ inserted: table containing new versions

# Triggers

Body 2: Loop through and reject updates

```
OPEN myRes;  
FETCH NEXT FROM myRes INTO @peakName,  
@oldHeight, @newHeight;  
  
WHILE (@@FETCH_STATUS = 0) BEGIN  
    PRINT 'You changed the height of ' + @peakName +  
        ' from ' + CAST (@oldHeight AS VARCHAR(100)) +  
        ' to ' + CAST (@newHeight AS VARCHAR(100)) +  
        ' . I am ignoring it .';  
    UPDATE peak SET elev = @oldHeight  
        WHERE name = @peakName;  
    FETCH NEXT FROM myRes INTO  
        @peakName,  
@oldHeight, @newHeight;  
END  
CLOSE myRes; DEALLOCATE myRes;
```

# Table Variables

No arrays, linked lists, etc. in T-SQL

▶ Encode everything as a table!

```
DECLARE @myMap TABLE (  
    myKey INTEGER,  
    myValue VARCHAR (200),  
    PRIMARY KEY (myKey);  
);
```

```
DECLARE @my2DArray TABLE (  
    xPos INTEGER,  
    yPos INTEGER,  
    value DOUBLE  
    PRIMARY KEY (xPos, yPos);  
);
```

# Temporary Tables

```
CREATE TABLE #myMap (  
    myKey INTEGER,  
    myValue VARCHAR (200),  
    PRIMARY KEY (myKey);  
);
```

When to use (vs. table variables)

- ▷ “We recommend using table variables instead of temporary tables.”
- ▷ Exception: when you want to give callees access
- ▷ Not possible to send table vars as params in TSQL 2005
- ▷ Can be done in newer versions!

Questions?