Stored Procedures

STORED QUERIES CALLED ON DEMAND!

Stored Procedures

Stored Procedures (SPs) are pre-written SQL queries stored by the DBMS

- They are similar to Views in that they are called when required
  - They (technically) do not store data
- Unlike views they may contain INSERT, UPDATE, DELETE, SELECT queries or any combination of these!
  - Often SPs are used to perform basic CRUD tasks (create, read, update, delete)
  - Other SPs are used to perform complex calculations needed for generating reports or transforming data

Stored Procedures - Creation

The syntax for creating a stored procedure is relatively simple

```
CREATE PROCEDURE GetSimpsons
AS

SELECT * FROM Characters
WHERE CharacterName LIKE '%Simpson'
```

Create Clause

SP Query(s)

Where have you seen “CREATE” before?

Stored Procedures - Execution

The syntax for executing a stored procedure is relatively simple

- This simply runs the queries contained in the stored procedure

```
EXECUTE GetSimpsons

-- OR

EXEC GetSimpsons
```


Results | Messages

	CharacterID	CharacterName	CharacterRole
1	5	Amber Simpson	Homer Simpson's
2	12	Bart Simpson	Eldest child and on
3	62	Grampa Abraham Simpson	Homer Simpson's f
4	69	Homer Simpson	Husband of Marge
5	92	Lisa Simpson	Middle child and el
6	98	Maggie Simpson	Youngest child and
7	100	Marge Simpson	Wife of Homer; mo
8	106	Mona Simpson	Estranged mother

Stored Procedures - Modification

The syntax for modifying a stored procedure is also relatively simple

```
ALTER PROCEDURE GetSimpsons
AS

SELECT CharacterID, CharacterName FROM Characters
WHERE CharacterName LIKE '%Simpson'
```

Use “ALTER” instead of “CREATE”!!!

Where have you seen “ALTER” before?

```
EXECUTE GetSimpsons

-- OR

EXEC GetSimpsons
```

Results | Messages

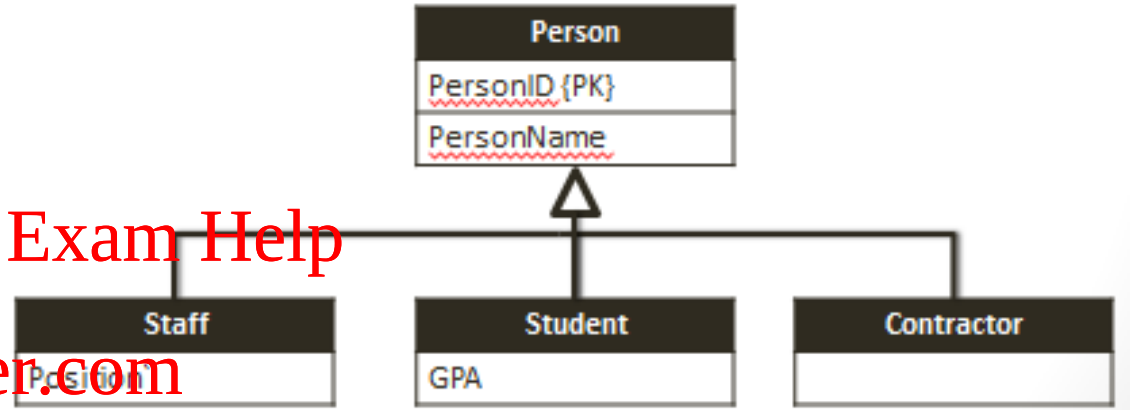
	CharacterID	CharacterName
1	5	Amber Simpson
2	12	Bart Simpson
3	62	Grampa Abraham Simpson
4	69	Homer Simpson
5	92	Lisa Simpson
6	98	Maggie Simpson
7	100	Marge Simpson
8	106	Mona Simpson

Stored Procedures

Stored Procedures can be used like VIEWs

```
CREATE PROCEDURE GetStaff
AS

SELECT PersonID, PersonName, Position
FROM Person AS P JOIN Staff AS S
ON P.PersonID = S.PersonID
```


```
CREATE VIEW StaffView
AS
SELECT PersonID, PersonName, Position
FROM Person AS P JOIN Staff AS S
ON P.PersonID = S.PersonID
```

```
EXECUTE GetStaff

-- VS

SELECT * FROM StaffView
```

Views Vs Stored Procedures

Unfortunately the similarities stop there ☹

- A View can readily be used in other queries
- A Stored Procedure cannot – it requires more work

JOIN Query using SP


```
DECLARE @tablevar TABLE (
charID int,
charName varchar(100)
);

INSERT INTO @tablevar(charID, charName) EXEC GetSimpsons

SELECT * FROM @tablevar AS t1 JOIN @tablevar AS t2
ON t1.charName = t2.charName
```


Same JOIN Query using View

```
SELECT V1.charID, V1.charName
FROM SimpsonsView AS V1
JOIN SimpsonsView AS V2
ON V1.charName = V2.charName
```

Views Vs Stored Procedures

Unfortunately the similarities stop there ☹

- Stored Procedures are more powerful than Views
- They are not the same thing…

Unlike a View, a stored procedure can:

- Contain flow control statements (IF, ELSE, IF ELSE)
- Contain many separate statements that work collectively or separately on data
- Make use of variables
  - @someVariableName

- Views cannot!

SQL Variables - @varName

Variables (@someName) are placeholders for unknown values

- The value of a variable is provided by the user when it is needed
- It must be **declared**
- Like any other attribute it must have a known data type
- It can have a **default** value

Single Declaration

```
DECLARE @someInt INT
```

Multiple Declaration

```
DECLARE    @someString VARCHAR(50),
           @someDate DATETIM
```

Multiple Declaration with Default values

```
DECLARE    @someString VARCHAR(50) = 'DefaultValue',
           @someDate DATETIME = GetDate()
```

SQL Variables - @varName

Variables (@someName) can be used to return values

```
DECLARE
        @someString VARCHAR(50) = 'DefaultValue',
        @someDate DATETIME = GetDate()

SELECT      @someString AS someText,
            @someDate AS currentDate
```


Results | Messages

	someText	currentDate
1	DefaultValue	2015-09-24 09:33:31.230

SQL Variables - @varName

Variables (@someName) can be have their values SET

```
DECLARE
        @someString VARCHAR(50) = NULL
        @someDate DATETIME = NULL

SET @someString = 'DefaultValue';
SET @someDate = GetDate();

-- VS

SELECT  @someString = 'DefaultValue',
                      @someDate = GetDate();
```



```
SELECT
        @someString AS someText,
        @someDate AS currentDate
```

Results | Messages

	someText	currentDate
1	DefaultValue	2015-09-24 09:33:31.230

SQL Variables - @varName

Variables (@someName) can be supplied values from queries

- Be careful the query only returns ONE row and not many!!

```
DECLARE
    @character VARCHAR(100) = NULL
    @episode VARCHAR(100)= NULL

SELECT      @character = characterName,
            @episode = episodeName
FROM Characters AS C JOIN Episodes AS E
ON C.EpisodeID = E.EpisodeID
WHERE CharacterID = 5

SELECT @character, @episode
```

NOTE: Not all DBMS allow use of SELECT for this purpose!!

Results | Messages

	(No column name)	(No column name)
1	Amber Simpson	"Viva Ned Flanders"

SQL Variables - @varName

Variables (@someName) can be supplied values from queries

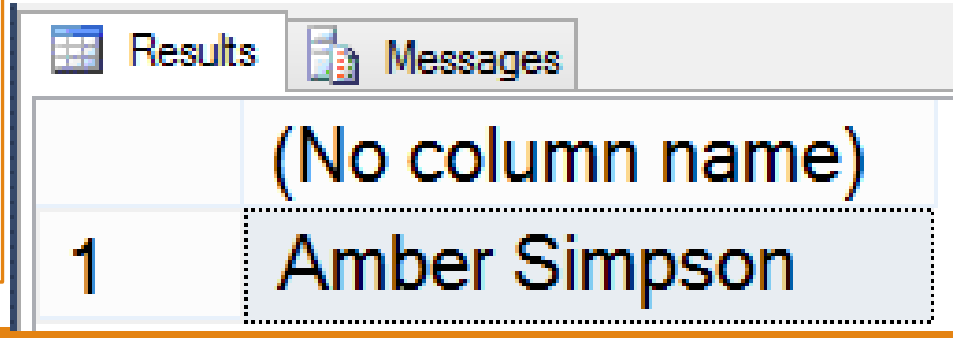
- Be careful the query only returns ONE row and not many!!

```
DECLARE
        @character VARCHAR(100) = NULL
        @episode VARCHAR(100)= NULL

SET@character = (
    SELECT characterName
    FROM Characters AS C JOIN Episodes AS E
    ON C.EpisodeID = E.EpisodeID
    WHERE CharacterID = 5
)
SELECT @character
```


SET is the standard clause and can be used to set the value of only one variable at a time!

The query in this case MUST return only ONE Row else an error will occur



SQL Variables and Stored Procedures

SQL Variables are often used in stored procedures

- They may be used to pass values to a SP query
- They may be used to return values from a SP query**
  - This is different from returning values as a result set/table


```
CREATE PROCEDURE GetCharacters
@Name VARCHAR(100)

AS

SELECT * FROM Characters WHERE
CharacterName LIKE @Name + '%'
```

Variable Declarations separated by ","

To run the query, execute with values

```
EXEC GetCharacters 'Homer'
```

SQL Variables and Stored Procedures

SQL Variables are often used in stored procedures

- Use them wisely!!!

```
CREATE PROCEDURE GetCharactersV2
@Name VARCHAR(100),
@Aired DATE
AS

SELECT * FROM Characters AS C
JOIN Episodes AS E
ON C.EpisodeID = E.EpisodeID
WHERE
CharacterName LIKE @Name + '%'
OR DateAired = @Aired
```



```
EXEC GetCharactersV2 'Homer', NULL
EXEC GetCharactersV2 'Homer', '1987-04-19'
```

SQL Variables and Stored Procedures

SQL Variables are often used in stored procedures

```
CREATE PROCEDURE InsertCharacter
@characterName VARCHAR(100),
@characterRole VARCHAR(100),
@characterID INT OUTPUT

AS

INSERT INTO Characters (CharacterName, CharacterRole)
VALUES (@characterName, @characterRole)

SELECT @characterID = SCOPE_IDENTITY()
RETURN
```

SQL Variables and Stored Procedures

SQL Variables are often used in stored procedures

- Getting the result back in SQL is bizarre!
  - You may rarely do this at all


```
ARE @newCharacterID INT

InsertCharacter   'Test',
                  'Test',
                  @characterID = @newCharacterID OU

CT @newCharacterID
```

Who ever thought of assigning the value of a variable backward!

SQL Variables and Stored Procedures

SQL Variables are often used in stored procedures

```
CREATE PROCEDURE SaveCharacter
@characterName VARCHAR(100),
@characterRole VARCHAR(100)

AS

DECLARE @characterID INT
SET @characterID = (
SELECT CharacterID FROM Characters
WHERE CharacterName = @characterName
)

-- Contd.. Next page
```

SQL Variables and Stored Procedures

Contd...

```
IF(@characterID  IS NULL)
BEGIN
INSERT INTO Characters (CharacterName, CharacterRole)
VALUES (@characterName, @characterRole)
END

ELSE
BEGIN
UPDATE Characters SET
CharacterRole = @characterRole
WHERE CharacterID = @characterID
END
```



```
EXEC GetCharactersV2 'Homer', NULL
EXEC GetCharactersV2 'Homer', `Father of Bart'
```

Stored Procedures

Stored Procedures are useful for many reasons:

1. Maintainability
   - Because the queries are in one location, updates and tracking of dependencies based on schema changes becomes easier
   - They can be used for storing standard INSERT, UPDATE, DELETE and SELECT Queries that are regularly used

2. They can hide complex queries
   - Queries cab be written and tested independent of an application that may need to use them

3. They can restrict access to confidential data held in specific columns
   - They can be used to limit direct access to tables in the database