COMP1140: Database and Information Management

Lecture Note – Week 10

Dr Suhuai Luo

School of EEC

University of Newcastle

Note

- Assignment 3 is due week 12
- You have to attend lab in week 12 to get your assignment 3 marked
- SQL test marking will be done by week 11.

Last week

- SQL test
  - Result will be given next week
  - Note: if you are allowed to do resit for the test, pls contact lecturer for detailed arrangement

This week

- Views
- Transactions
- Triggers
- Stored Procedures
- Issues and discussions and feedback of A2 & A3
- Reference:
  - Views – Section 7.4
  - Transactions – Section 7.5
  - Triggers – Chapter 8.3, 29.4.12
  - Stored Procedures – Section 8.2

Views

- **View**: A virtual relation that does not actually exist in the database but is produced upon request, at time of request
- Contents of a view are defined as a query on one or more base relations
- Any operations on views are automatically translated into operations on relations from which the views are derived

Views (cont'd)

- Purpose of a view:
  - simplify query commands (especially simplify multi table query)
  - Provide valuable data security
  - Significantly enhance programming consistency & productivity for database
- In SQL Server, views are at the same level as tables

SQL - CREATE VIEW

- Creating views:

```
CREATE VIEW ViewName [ (newColumnName [,...]) ]
  AS subselect
  [WITH CHECK OPTION]
```

- Can assign a name to each column in view.
- If list of column names is specified, it must have same number of items as number of columns produced by *subselect*.
- If list omitted, each column takes name of corresponding column in *subselect*.

SQL - CREATE VIEW (contd.)

```
CREATE VIEW ViewName [ (newColumnName [,...]) ]
 AS subselect
 [WITH CHECK OPTION]
```

- The list must be specified if there is any ambiguity in a column name.
- The *subselect* is known as the <u>defining query</u>.
- WITH CHECK OPTION ensures that if a row fails to satisfy WHERE clause of defining query, it is not added to underlying base table of the view.

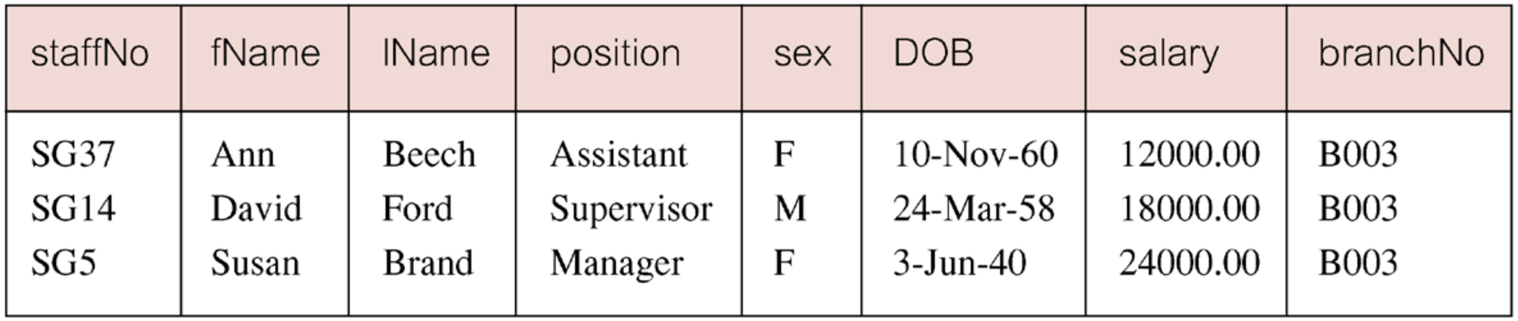
Example: Create Horizontal View

- Create view so that manager at branch B003 can only see details for staff who work in his or her branch.

```
CREATE VIEW Manager3Staff
AS      SELECT *
        FROM Staff
        WHERE branchNo = ‘B003’;
```

- Run: select * from Manager3Staff

Table 6.3 Data for view Manager3Staff.



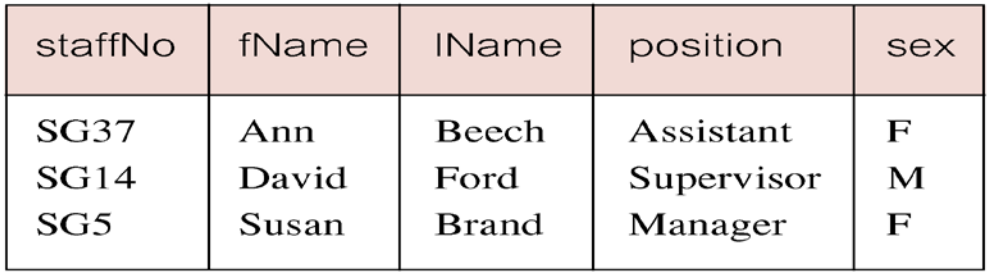
staffNo	fName	lName	position	sex	DOB	salary	branchNo
SG37	Ann	Beech	Assistant	F	10-Nov-60	12000.00	B003
SG14	David	Ford	Supervisor	M	24-Mar-58	18000.00	B003
SG5	Susan	Brand	Manager	F	3-Jun-40	24000.00	B003

Example: Create Vertical View

- Create view of staff details at branch B003 excluding DOB, salary, BrachNo.

```
CREATE VIEW Staff3
AS  SELECT staffNo, fName, lName, position, sex
    FROM Staff
    WHERE branchNo = ‘B003’;
```

Table 6.4 Data for view Staff3.



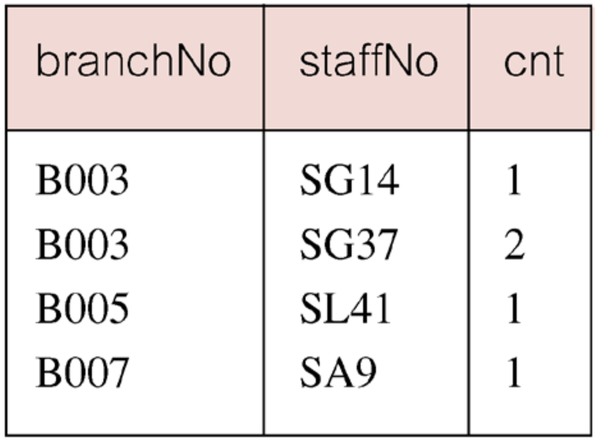
staffNo	fName	lName	position	sex
SG37	Ann	Beech	Assistant	F
SG14	David	Ford	Supervisor	M
SG5	Susan	Brand	Manager	F

Example: Grouped and Joined Views

Create view of staff who manage properties for rent, including branch number they work at, staff number, and number of properties they manage.

```
CREATE VIEW StaffPropCnt (branchNo, staffNo, cnt)
 AS SELECT s.branchNo, s.staffNo, COUNT(*)
        FROM Staff s, PropertyForRent p
        WHERE s.staffNo = p.staffNo
        GROUP BY s.branchNo, s.staffNo;
```

Table 6.5 Data for view StaffPropCnt.



branchNo	staffNo	cnt
B003	SG14	1
B003	SG37	2
B005	SL41	1
B007	SA9	1

SQL - DROP VIEW

- DROP VIEW causes definition of view to be deleted from database.

```
DROP VIEW ViewName [RESTRICT | CASCADE]
```

- For example:

```
DROP VIEW Manager3Staff;
```

- With CASCADE, all related dependent objects are deleted; i.e. any views defined on the view be dropped.
- With RESTRICT (default), if any other objects depend for their existence on view being dropped, command is rejected

View Updatability

- All updates to base table are reflected in all views that encompass the base table.
- Similarly, may expect that if view is updated then base table(s) will reflect the change.
- However, views are not always updatable!!!

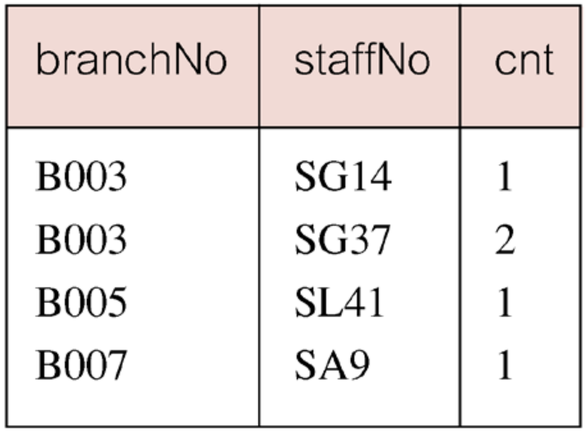
Example 1: View Updatability

- Consider the following insert on view StaffPropCnt.

```
INSERT INTO StaffPropCnt
VALUES (‘B003’, ‘SG5’, 2);
```

- Need to insert 2 records into PropertyForRent. Primary keys are unknown!!!

Table 6.5 Data for view StaffPropCnt.



branchNo	staffNo	cnt
B003	SG14	1
B003	SG37	2
B005	SL41	1
B007	SA9	1

Example 2: View Updatability

- Consider the following view:

```
CREATE VIEW StaffPropList (branchNo, staffNo,
                                        propertyNo)
AS SELECT s.branchNo, s.staffNo, p.propertyNo
    FROM Staff s, PropertyForRent p
    WHERE s.staffNo = p.staffNo;
```

- Now try to insert the record:

```
INSERT INTO StaffPropList
VALUES (‘B003’, ‘SG5’, ‘PG19’);
```

Example 2: View Updatability (contd.)

- In PropertyForRent all columns except postcode/staffNo are not allowed nulls.
- No way of giving remaining non-null columns values.

ISO: View Updatability

- ISO specifies that a view is updatable if and only if:
  - DISTINCT is not specified.
  - Every element in SELECT list of defining query is a column name (rather than constant, expression or aggregate) and no column appears more than once.
  - FROM clause specifies only one table, excluding any views based on a join, union, intersection or difference.
  - No nested SELECT referencing outer table.
  - No GROUP BY or HAVING clause in defining query.
  - Also, every row added through view must not violate integrity constraints of base table.

Updatable View

- For view to be updatable, DBMS must be able to trace any row or column back to its row or column in the source table.

- Many DBMSs allow only updates on views based on a single relation!

WITH CHECK OPTION

- Rows exist in a view because they satisfy WHERE condition of defining query.
- If a row changes and no longer satisfies condition, it disappears from the view.
- WITH CHECK OPTION prohibits a row being changed via the view which results the row disappearing from the view.

Example: WITH CHECK OPTION

```
CREATE VIEW Manager3Staff
  AS       SELECT *
           FROM Staff
           WHERE branchNo = 'B003'
  WITH CHECK OPTION;
```

- Consider the following operations
  - INSERT INTO Manager3Staff (staffNo, branchNo) VALUES ('SG10', 'B002');

  -
    ```
    UPDATE   Manager3Staff
    SET      branchNo = 'B002'
    WHERE    staffNo = 'SG37'
    ```

NOT ALLOWED: Changing the branchNo to B002 will make the row(s) out of view

Advantages of Views

- **Data independence**: hide changes made to underlying base tables
- **Currency**: up-to-date values from base tables
- **Improved security**: provide access to necessary data to users, hiding unnecessary data
- **Reduced complexity, Convenience, and Customization**: complex queries can be hidden in a view, simpler (related-only) queries asked from view.
- **Data integrity**: WITH CHECK OPTION, ensures that disallowed updates are not performed

Disadvantages of Views

- **Update restriction**: Not all views are updatable

- **Performance**: Queries on complex views may be slower than directly asking from base tables as a translation of query to base table is required.

Transaction Basics

- A logical unit of work consisting of one or more SQL statements
- Certain actions on a database needs to be performed atomically
- Transactions in database provide a means to perform actions atomically, consistently and reliably by multiple users

Transactions Commands in SQL

- BEGIN TRANSACTION:
  - Initiates a transaction
- COMMIT TRANSACTION:
  - Completes the transaction
- ROLLBACK TRANSACTION:
  - Cancels the transaction

Properties of Transactions

- ACID properties:
  - **A**tomicity: Every transaction is an atomic operations (no-partial transactions)
  - **C**onsistency: Every transaction, if started in a consistent state, results in a consistent state of the database
  - **I**solation: Every transaction is isolated from the effects of other concurrently executing transactions
  - **D**urability: A transaction is durable in face of system failures.

Atomicity

- Every transaction must occur as a single unit of work (“all or nothing”).

- Example,
  - Transferring $100 from Account A to Account B
  - This requires 2 update statements on table Account (assume a table Account exists with accNo and balance)

Atomicity (contd.)

Both these statements must execute as a **single** unit. If **only** 1st update is executed **without** the 2nd update, lose money!!!

- Example (contd.)
  - UPDATE Account
    SET balance = balance – 100
    WHERE accNo = ‘A’
  - UPDATE Account
    SET balance = balance + 100
    WHERE accNo = ‘B’

```
UPDATE      Account
SET         balance = balance – 100
WHERE       accNo = ‘A’

UPDATE      Account
SET         balance = balance + 100
WHERE       accNo = ‘B’
```

Consistency

- A single transaction, if it begins with a consistent database state, always completes in a consistent state.
- Consistency is maintained by the DBMS by satisfying all integrity constraints
- For example, in the Account table, if there was a CHECK constraint CHECK (balance >= 0) then, if transaction successfully completes, balance >= 0 will hold for account balances

Isolation

- When multiple transactions work on the same data, non-isolation may result in incorrect data.

- Example,
  - Transaction 1: Transfers $100 from Account A to Account B
  - Transaction 2: Calculates 10% interest on all accounts for their current balance

Total interest paid by bank = $**190**
Total interest that should be paid = $**200**
Customer lost $**10** interest!

Example: Isolation

Transaction 1	**Transaction 2**	**Account A Balance**	**Account B Balance**
		1000	1000

UPDATE Account
SET balance = balance – 100
WHERE accNo = 'A'			

		900	1000

	UPDATE Account
	SET balance = balance * 1.1		

		990	1100

UPDATE Account
SET balance = balance + 100
WHERE accNo = 'B'			

		990	1200

Example: Isolation

- When multiple transactions work on the same data and at least one is an update, this can result in conflict that causes inconsistencies.

- DBMS must ensure that the each transaction is isolated from the effects of concurrently executing transactions.

Transaction Isolation Levels

- ISO defines different isolation levels which ensures the degree of isolation of transactions.
- Isolation level SERIALIZABLE ensures that the transaction is completely isolated from other concurrently executing transactions
- Example
  - SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

Durability

- Durability means that successfully completed (i.e. committed) transactions are permanently recorded in DB and must not be lost because of a subsequent system failure.

Example: SQL Transaction

- Transfer funds example:

```
SET TRANSACTION ISOLATION LEVEL SERIALIZABLE

BEGIN TRANSACTION

    UPDATE      Account
    SET         balance = balance – 100
    WHERE       accNo = ‘A’

    UPDATE      Account
    SET         balance = balance + 100
    WHERE       accNo = ‘B’

COMMIT TRANSACTION
```

Programmatic Extensions in SQL:1999 and SQL:2003

- In SQL:1999 and SQL:2003, a number of new statements were added to make the language computationally complete.
- SQL was mainly a declarative language but with these extensions, programming capabilities are added to SQL.
- Note that there are differences in support for these statements in different dialects (such as T-SQL).

* We will learn a small subset of these extensions!

Declaring Variables

- Statements can be grouped together into a compound statement (block) with its own local variables
- Example: Declaring local variables
  - DECLARE @i INTEGER;
- An assignment statement allows the result of a SQL value to be assigned to local variable
  - @i = 10;

IF statement

- The IF … THEN … ELSE … END IF statement allows conditional processing

- Example:

```
IF position = 'Manager' THEN
          RETURN TRUE;
ELSE
          RETURN FALSE;
END IF
```

WHILE Loop

- A set of statements allows for repeated execution of a block of SQL statements

- Example

```
WHILE b <> TRUE DO
  ...
END WHILE
```

WHILE Loop (contd.)

- BREAK

  Causes an exit from the innermost WHILE loop. Any statements appearing after the END keyword, marking the end of the loop, are executed.

- CONTINUE

  Causes the WHILE loop to restart, ignoring any statements after the CONTINUE keyword.

Triggers

- It is an SQL (compound) statement executed **automatically** by DBMS as side effect of a modification to named table.
- Use of triggers include:
    - Validating input data and maintaining complex integrity constraints that otherwise would be difficult/impossible.
    - Supporting alerts.
    - Maintaining audit information.
    - Supporting replication.

Triggers (contd.)

```
CREATE TRIGGER TriggerName
BEFORE | AFTER <triggerEvent>
        ON <TableName>
[REFERENCING <oldOrNewValuesAliasList>]
[FOR EACH {ROW | STATEMENT}]
[ WHEN (triggerCondition) ]
<triggerBody>
```

Triggers (contd.)

- BEFORE - trigger is fired before (AFTER - trigger is fired after) associated event occurs.
- <triggerEvent> could be an INSERT, UPDATE or DELETE statement which cause the trigger to fire (i.e. execute). These types of trigger are also known as DML triggers.

- ON <TableName> is the table on which the trigger is defined
- (triggerCondition): condition for trigger to fire (e.g., xxx.yyy is NULL)

Triggers (contd.)

- Triggered action is SQL procedure statement, which can be executed in one of two ways:
  - For each row (FOR EACH ROW) affected by the event. This is called a row-level trigger.
  - Only once for entire event (FOR EACH STATEMENT), which is default. This is called a statement-level trigger.

- <oldOrNewValuesAliasList>refers to
  - An old or new row (OLD/NEW or OLD ROW/NEW ROW)
  - An old or new table (OLD TABLE/NEW TABLE)

- Clearly, old values are not applicable for INSERT events, and NEW values not for DELETE events

Example: Use of AFTER Trigger

```
CREATE TRIGGER InsertMailshotTable
    AFTER INSERT ON PropertyForRent
    REFERENCING NEW ROW AS pfr
    BEGIN
            INSERT INTO Mailshot VALUES
            (SELECT c.fName, c.lName, c.maxRent,
             pfr.propertyNo, pfr.street, pfr.city, pfr.postcode,
             pfr.type, pfr.rooms, pfr.rent
            FROM Client c, pfr
            WHERE c.branchNo = pfr.branchNo AND
            (c.prefType=pfr.type AND c.maxRent <= pfr.rent) )
    END;
```

Triggers - Advantages and Disadvantages

- Major advantage - standard functions can be stored within database and enforced consistently.
- This can dramatically reduce complexity of applications.
- However, there can be some disadvantages:
  - Complexity.
  - Hidden functionality.
  - Performance overhead.

T-SQL Stored Procedures

- What is a stored procedure?
  - Similar to a procedure in any other programming language, it is a procedure executed by the Database Engine

- SQL Server's stored procedures:
  - Stored procedures written in T-SQL
  - Stored procedure in .NET language (CLR stored procedures)
  - System stored procs
  - etc.

T-SQL Stored Procedures (contd.)

- Examples –
  - Simple Store procedure
  - Passing in values
  - Passing out values

T-SQL Stored Procedures (contd.)

- A simple T-SQL stored procedure

```
-- Creating stored proc
CREATE PROCEDURE PrintAllStudent
AS
    SELECT *
    FROM Student
GO

-- Calling stored proc
EXECUTE PrintAllStudent
```

T-SQL Stored Procedures (contd.)

- A stored procedure with input parameters

```
-- Creating stored proc
CREATE PROCEDURE PrintStudentInfo @stdNo CHAR(5)
AS
    SELECT *
    FROM Student
    WHERE stdNo = @stdNo
GO

-- Calling stored proc
EXECUTE PrintStudentInfo 'S0001'
```

T-SQL Stored Procedures (contd.)

- A stored procedure with input parameters and output parameter

```
CREATE PROCEDURE NoCoursesRegisteredByStudent
   @stdNo CHAR(5), @noCourses INT OUTPUT
AS
    SELECT @noCourses = COUNT(*)
    FROM Register
   WHERE stdNo = @stdNo
GO
```

T-SQL Stored Procedures (contd.)

```
-- Calling stored proc
DECLARE @nCourses INT
EXECUTE NoCoursesRegisteredByStudent 'S0001',
   @nCourses OUT
PRINT 'Number of courses registered by S0001 = ' +
   CAST(@nCourses AS CHAR)
```

- Dropping stored procedure

```
DROP PROCEDURE NoCoursesRegisteredByStudent
```

Summary

- Views
- Transactions
- Triggers
- Stored Procedures

Lab This Week

- Review and practice on View, Transaction, Trigger

Issues and feedbacks on Assignment 2

- Comments on A2
- You need to revise content of A2, then work on A3

Issues and discussions on Assignment 3

- A3 specification
- A3marking guide