CSE 4020/5260
Database Systems

Instructor: Fitzroy Nembhard, Ph.D.

Week 11
Advanced SQL

FLORIDA TECH
FLORIDA'S STEM UNIVERSITY®

Distribution

- All slides included in this class are for the exclusive use of students and instructors associated with Database Systems (CSE 4020/5260) at the Florida Institute of Technology
- Redistribution of the slides is not permitted without the written consent of the author.

Advanced SQL

- Top-N Queries
- Assertions
- Triggers
- Stored Procedures
- Embedded & Dynamic SQL
- ODBC & JDBC

Top-N Queries

- Some systems (including MySQL and PostgreSQL) allow a clause LIMIT N to be added at the end of an SQL query to specify that only the first n tuples should be output

```
select ID, GPA
from student
order by GPA desc
limit 10;
```

Top-N Queries (Cont'd)

- Oracle (both current and older versions) offers the concept of a row number to provide this feature. A special, hidden attribute rownum numbers tuples of a result relation in the order of retrieval.

```
select *
from (select ID, GPA
            from student_grades
            order by GPA desc)
 where rownum <= 10;
```

Assertions

- An *assertion* is a predicate expressing a condition that we wish the database always to satisfy.
- Similar to DDL check constraints, but they can test conditions across multiple tables.
- When an assertion is made, the system tests it for validity, and tests it again on every update that may violate the assertion.

Assertion Example 1

“The sum of all loan amounts for each branch must be no greater than the sum of all account balances at the branch.”

```
create assertion sum-constraint check
    (not exists (select * from branch
                 where (select sum(amount) from loan
                        where loan.branch-name =  branch.branch-name)
                     > (select sum(balance) from account
                        where account.branch-name = branch.branch-name)))
```

Assertion Example 2

"Every loan has at least one borrower who maintains an account with a minimum balance of $1000.00"

```
create assertion balance-constraint check
    (not exists (
         select loan-number from loan
         where not exists (
              select borrower.customer-name from borrower, depositor, account
              where loan.loan-number = borrower.loan-number
                    and borrower.customer-name = depositor.customer-name
                    and depositor.account-number = account.account-number
                    and account.balance >= 1000)))
```

Schema
branch (branch-name, branch-city, assets)
customer (customer-name, customer-street, customer-city)
account (account-number, branch-name, balance)
loan (loan-number, branch-name, amount)
depositor (customer-name, account-number)
borrower (customer-name, loan-number)

Triggers

- A *trigger* is a statement that is executed automatically by the system as a side effect of a modification to the database.

- A trigger has two parts:
  - conditions
  - actions

Abbreviated Trigger Syntax

```
create trigger [trigger_name]
[before | after]
{insert | update | delete}
on [table_name]
[for each row]
[trigger_body]
```

Destroying a Trigger

```
DROP TRIGGER
schema_name.trigger_name;
```

MySQL Trigger Syntax

- Note that you cannot associate a trigger with a temporary table or a view.

```
CREATE
    [DEFINER = user]
    TRIGGER trigger_name
    trigger_time trigger_event
    ON tbl_name FOR EACH ROW
    [trigger_order]
    trigger_body
```

Key

```
trigger_time: { BEFORE | AFTER }
trigger_event: { INSERT | UPDATE | DELETE }
trigger_order: { FOLLOWS | PRECEDES } other_trigger_name
```

What Fields are Available to Me in A Trigger?

- **INSERT TRIGGER**
  - Access to the **NEW** pseudo rows only.
- **UPDATE TRIGGER**
  - Access to the **NEW** and **OLD** pseudo rows
- **DELETE TRIGGER**
  - Access only to the **OLD** pseudo rows

Trigger Example

```
CREATE TRIGGER upd_check
AFTER UPDATE
ON some_table
FOR EACH ROW
BEGIN
    IF (OLD.last_changed_date <> NEW.last_changed_date)
        THEN
            INSERT INTO audit_table(ID, last_changed_date)
            VALUES (OLD.ID, OLD.last_changed_date);
    END IF;
END;
```

Trigger Example

- Suppose the bank deals with overdrafts by:
  - Setting the account balance to zero
  - Creating a loan in the amount of the overdraft
- Condition:
  - update to the account relation that results in a negative balance.
- Actions:
  - Create a loan tuple
  - Create a borrower tuple
  - Set the account balance to 0

Trigger Example 2 in MySQL

Schema

branch (branch-name, branch-city, assets)
customer (customer-name, customer-street, customer-city)
account (account-number, branch-name, balance)
loan (loan-number, branch-name, amount)
depositor (customer-name, account-number)
borrower (customer-name, loan-number)

```
create trigger overdraft-trigger after update on account
for each row
begin
   if NEW.balance < 0 then
         insert into loan values
            (NEW.account-number, NEW.branch-name, – NEW.balance);
         insert into borrower
            (select depositor.customer-name, depositor.account-number
             from depositor
             where NEW.account-number = depositor.account-number);
         update account set balance = 0
            where account.account-number = NEW.account-number
   end if;
end
```

MySQL Trigger Syntax

- MySQL has more limited trigger capabilities
  - Trigger execution is only governed by events, not conditions
    - Workaround: Enforce the condition within the trigger body
  - Old and new rows have fixed names: **OLD, NEW**
- Change the overdraft example slightly:
  - Also apply an overdraft fee!
- What if the account is already overdrawn?
  - Loan table will already have a record for overdrawn account…
  - Borrower table will already have a record for the loan, too!
  - The previous version of trigger would cause duplicate key error!

MySQL INSERT Enhancements

- MySQL has several enhancement to the INSERT command
  - (Most databases provide similar capabilities)
- Try to insert a row, but if key attributes are same as another row, simply don't perform the insert:
  - **`INSERT IGNORE INTO tbl ...;`**
- Try to insert a row, but if key attributes are same as another row, update the existing row:
  - **`INSERT INTO tbl ... ON DUPLICATE KEY UPDATE attr1 = value1, ...;`**
- Try to insert a row, but if key attributes are same as another row, replace the old row with the new row
  - If key is not same as another row, perform a normal **`REPLACE INTO tbl ...;`**

MySQL INSERT Enhancements

```
DELIMITER //
CREATE TRIGGER trigger_overdraft BEFORE UPDATE ON account
FOR EACH ROW
BEGIN
    DECLARE overdraft_fee NUMERIC(12, 2) DEFAULT 30;
    DECLARE overdraft_amt NUMERIC(12, 2);
         -- If an overdraft occurred then handle by creating/updating a loan.
         IF NEW.balance < 0 THEN
         -- Remember that NEW.balance is negative.
                  SET overdraft_amt = overdraft_fee - NEW.balance;

                  INSERT INTO loan (loan_number, branch_name, amount)
                  VALUES (NEW.account_number, NEW.branch_name, overdraft_amt)
                  ON DUPLICATE KEY UPDATE amount = amount + overdraft_amt;

                  INSERT IGNORE INTO borrower (customer_name, loan_number)
                  SELECT customer_name, account_number FROM depositor
                  WHERE depositor.account_number = NEW.account_number;

                  SET NEW.balance = 0;
         END IF;
END//
DELIMITER ;
```

Note that you need to remove the lines with the keyword "DELIMITER" when executing your trigger via code. Remove also the //.

FLORIDA TECH

Useful MySQL Trigger Tips

- **Defining and Assigning a Variable**

```
DECLARE overdraft_amt NUMERIC(12, 2);
SET @overdraft_amt := 35.0;
```

- **Storing the Results of a Query in a Variable**

```
SELECT (COUNT(*) INTO @total_count  FROM account);
```

Other MySQL Trigger Examples

```
CREATE TRIGGER amount_sum BEFORE INSERT ON account
FOR EACH ROW
SET @sum = @sum + NEW.amount;
```

```
CREATE TRIGGER totals_transaction BEFORE INSERT ON account
FOR EACH ROW PRECEDES amount_sum
SET
@deposits = @deposits + IF(NEW.amount>0,NEW.amount,0),
@withdrawals = @withdrawals + IF(NEW.amount<0,-NEW.amount,0);
```

```
CREATE TABLE test_table1(amount INT);
CREATE TABLE test_table2(amount INT);

CREATE TRIGGER test BEFORE INSERT ON test_table1
FOR EACH ROW
BEGIN
   INSERT INTO test_table2 values(…);
   SET @val = NEW.amount;
   DELETE FROM test_table WHERE amount = NEW.amount;
   UPDATE some_table SET column = column + 1 WHERE column = NEW.amount;
END;
```

FLORIDA TECH

Triggering Events and Actions in SQL

- Triggering event:
  - **insert**, **delete** or **update.**
- Triggers on update can be restricted to specific attributes:
  - **create trigger** *overdraft-trigger* **after update of** *balance* **on** *account*
- Values of attributes before and after an update can be referenced
  - **referencing old row as** (deletes and updates)
  - **referencing new row as** (inserts and updates)

When Not To Use Triggers

- Triggers, along with all the other integrity checking mechanisms, provide yet another opportunity to…slow up the database…

- Triggers can be used for many things:
  - Maintaining summary or derived data (e.g. total salary of each department).
  - Replicating databases.

- DBMSs have better, more efficient ways to do many of these things:
  - Materialized views - maintain summary data.
  - Data warehousing - maintaining summary/derived data.
  - Built-in support for replication.

Procedural Extensions and Stored Procedures

- SQL provides a **module** language that permits definition of procedures:
  - Conditional (if-then-else) statements
  - Loops (for and while)
  - Procedure definition with parameters
  - Arbitrary SQL statements

- Stored Procedures:
  - Stored in the DBMS.
  - Executed by calling them by name, on the command-line or from a program.
  - Permit external applications to operate on the database without knowing about internal details about the database or even SQL.
  - A standard that is not uncommon – put all queries in stored procedures; applications are then only allowed to call stored procedures.
  - In the simplest case, a stored procedure simply contains a single query.

Procedural Extensions and Stored Procedures Cont’d

- SQL Server Example: Return a set of authors

```
DELIMITER  //
CREATE PROCEDURE stpgetauthors
   @surname varchar(30)=null
   AS
BEGIN
   IF @surname = null
      BEGIN
         RAISERROR( 'No selection criteria provided !', 10, 1)
      END
   ELSE
      BEGIN
         SELECT * FROM authors
         WHERE au_lname LIKE @surname
      END
END //
DELIMITER ;
```

Mr. DB Here! In MySQL, you may show a message in the WorkBench using the following syntax:

```
IF Condition THEN
   SIGNAL SQLSTATE '45000’
   SET MESSAGE_TEXT = ‘Your Message’;
END IF;
```

Where ‘45000’ is is a generic SQLSTATE value that illustrates an unhandled user-defined exception

Procedural Extensions and Stored Procedures Cont'd

- Example: Return the capital for a state

```
DELIMITER //
CREATE PROCEDURE state_capital
(IN user_state CHAR(50))
BEGIN
  SELECT capital FROM us_states
  WHERE state = user_state;
END //
DELIMITER ;
```

Procedure Syntax: Notice the mix of inout/in/out variables

```
DELIMITER //
CREATE PROCEDURE PROC_NAME(IN VAR1 data_type1,
                           INOUT VAR2 data_type2,
                           …
                           OUT VARN data_typen)
BEGIN
  SQL STATEMENTS
END //
DELIMITER ;
```

Procedural Extensions and Stored Procedures Cont'd

- Calling a stored Procedure in Python

```
from mysql.connector import MySQLConnection
from python_mysql_dbconfig import read_db_config

def call_state_capital():
    try:
        db_config = read_db_config()
        conn = MySQLConnection(**db_config)
        cursor = conn.cursor()

        cursor.callproc('some_stored_procedure')

        # print out the result
        for result in cursor.stored_results():
            print(result.fetchall())

    except Error as e:
        print(e)

    finally:
        cursor.close()
        conn.close()
```

State Capital Example

```
args = ['Florida']
procedure_result = cursor.callproc('state_capital', args)
print(procedure_result [1])
```

Follow the tutorial here: https://www.mysqltutorial.org/python-connecting-mysql-databases/ to learn how to use MySQL config files for usernames, etc..

Procedural Extensions and Stored Procedures Cont'd

- Calling a stored Procedure in Java

```
import java.sql.*;

public class ProcedureCaller {
    public void callStateCapital(String stateName) {
        try {
            Connection conn = DriverManager.getConnection(DB, USER, PASS);
            CallableStatement statement = conn.prepareCall("{call state_capital(?)}"); // 1 parameter, so 1 wildcard

            statement.setString(1, stateName);
            statement.execute();

            // retrieve the result from the procedure
            String stateCapital = statement.getString(1);
            System.out.println("The capital for " + stateName + " is " + stateCapital);
            statement.close();
            conn.close();


        } catch (SQLException e) {
            System.err.println("An error occurred while trying to call a procedure");
            System.err.println(e.getMessage());
        }
    }
}
```

Submitting Queries from Programs

- Programmatic access to a relational database:
  - Embedded SQL
  - Dynamic SQL

- Standards for Dynamic SQL:
  - ODBC
  - JDBC

Oracle Embedded SQL Example

```
#include <stdio.h>
exec sql include sqlca;

char user_prompt[] = "Please enter username and password:  ";
char cid_prompt[] = "Please enter customer ID:  ";

int main()
{
    exec sql begin declare section;       /* declare SQL host variables    */
        char cust_id[5];
        char cust_name[14];
        float cust_discnt;                 /* host var for discnt value    */
        char user_name[20];
    exec sql end declare section;

    exec sql whenever sqlerror goto report_error; /* error trap condition     */
    exec sql whenever not found goto notfound; /* not found condition      */

    exec sql unix:postgresql://csc4380.cs.rpi.edu/sibel AS myconnection USER :user_name;
    /* ORACLE format: connect  */

    while (prompt(cid_prompt, 1, cust_id, 4) >= 0) {
        exec sql select cname, discnt
                into :cust_name, :cust_discnt   /* retrieve cname, discnt   */
                from customers where cid = :cust_id;
        exec sql commit work;                  /* release read lock on row */

        printf("CUSTOMER'S NAME IS  %s AND DISCNT IS  %5.1f\n",
             cust_name, cust_discnt);           /* NOTE, (:) not used here  */
        continue;
    }
}
```

Source: https://www.cs.rpi.edu/~sibel/csci4380/fall2019/course_notes/sql_procedural2.html

SQL Server Dynamic SQL Example

```
DECLARE @dynamic_sql NVARCHAR(max)
select @dynamic_sql = 'select * from orders'
EXEC(@dynamic_sql)
```

ODBC

- The Microsoft Open Database Connectivity (ODBC) interface is a C programming language interface that makes it possible for applications to access data from a variety of database management systems (DBMSs)

- ODBC defines an API providing the functionality to:
  - Open a connection to a database
  - Execute queries and updates
  - Get back results

ODBC (Cont.)

- An ODBC program first allocates an “SQL environment,” and then a “database connection handle.”
- An ODBC program then opens the database connection using SQLConnect() with the following parameters:
  - connection handle
  - server to connect to
  - userid
  - password
- Must also specify types of arguments:
  - SQL_NTS denotes previous argument is a null-terminated string.

ODBC Example Code

```
#include <stdio.h>
#include <sql.h>
#include <sqlext.h>

int ODBCexample()
  {
 HENV     env;     /* environment */
 HDBC     conn;    /* database connection */
 SQLAllocEnv(&env);
 SQLAllocConnect(env, &conn);
 SQLConnect(conn,
           "aura.bell-labs.com", SQL_NTS,
           "avi", SQL_NTS, "avipasswd", SQL_NTS);

{ //…. Do actual work … }

 SQLDisconnect(conn);
 SQLFreeConnect(conn);
 SQLFreeEnv(env);
 }
```

ODBC Code (Cont.)

- Main body of program (i.e., “Do actual work”):

```
char branchname[80];
float balance;
int    lenOut1, lenOut2;
HSTMT  stmt;
RETCODE error;   /* query return code */

SQLAllocStmt(conn, &stmt);
char* sqlquery = "select branch_name, sum (balance)
                          from account
                          group by branch_name";
error = SQLExecDirect(stmt, sqlquery, SQL_NTS);
if (error == SQL_SUCCESS) {
     SQLBindCol(stmt, 1, SQL_C_CHAR,   branchname , 80, &lenOut1);
     SQLBindCol(stmt, 2, SQL_C_FLOAT, &balance,         0 , &lenOut2);
         while (SQLFetch(stmt) >= SQL_SUCCESS) {
         printf (" %s  %g\n", branchname, balance);
     }
}
SQLFreeStmt(stmt, SQL_DROP);
```

JDBC

- JDBC is a Java *specific* API for communicating with database systems supporting SQL.
- JDBC supports a variety of features for querying and updating data, and for retrieving query results.
- Similar to ODBC in general structure and operation:
  - Open a connection
  - Create a “statement” or PreparedStatement object
  - Execute queries using the Statement object to send queries and fetch results
  - Exception mechanism to handle errors

JDBC

- Note that JDBC was introduced in Week 3 as sample code for Java programmers to connect to AWS.
- Example connection Strings:
  - DB_URL = "jdbc:mysql://localhost:3306/databaseName?characterEncoding=utf8";
  - DB_URL = "jdbc:mysql://drfitz.coolprofessor.us-east-1.rds.amazonaws.com/cse4020";
- Example Code:

```
public static void main(String[] args) {
    try
    {
        // Open a connection
        Connection conn = DriverManager.getConnection(DB_URL, USER, PASS);
        Statement stmt = conn.createStatement();
        String sql = "CREATE TABLE student " +
                "(ID varchar(5), " +
                " name varchar(20), " +
                " dept_name varchar(20), " +
                " tot_cred numeric(3,0) check(tot_cred >=0), " +
                " PRIMARY KEY ( ID ))";
        //Execute the DDL Statement
        stmt.executeUpdate(sql);
        System.out.println("Table created successfully in the database...");
    } catch (SQLException e) {
        e.printStackTrace();
    }
}
```