CMSC 508
Databases

Advanced SQL (I)

VCU
College of Engineering

Dr. Alberto Cano
Associate Professor
Department of Computer Science

Chapter 3 from Database System Concepts, 7th Ed. by Silberschatz, Korth, Sudarshan, 2019
Chapter 5 from Database Management Systems, 3rd Ed. by Ramakrishnan, Gehrke, 2003

- Calculated columns

- columnname datatype [GENERATED ALWAYS] AS (expression) [VIRTUAL | STORED]

```
CREATE TABLE tablename (
        first_name VARCHAR(50),
        last_name VARCHAR(50),
        fullname VARCHAR(101) AS (CONCAT(first_name,' ',last_name)) STORED,
);
```

- **MySQL** does **not** allow a volatile function such as curdate() in a generated column (e.g. cannot calculate date based on birthdate)

- Syntax and functionality allowed is limited. VIEWs will resolve this issue.

- Referential actions

```
CREATE TABLE tablename (
        …
        FOREIGN KEY (col) REFERENCES table (col)    [ON UPDATE action]
                                                    [ON DELETE  action]
);
```

- Referential actions: CASCADE, SET NULL, SET DEFAULT
- **CASCADE**: Delete or update the row from the parent table, and **automatically** delete or update the matching rows in the child table. **CAREFUL** with on delete cascade.
- **SET NULL**: matching rows in the child table will be set to NULL.
- **SET DEFAULT**: matching rows in the child table will be set to the default value.

- Referential actions

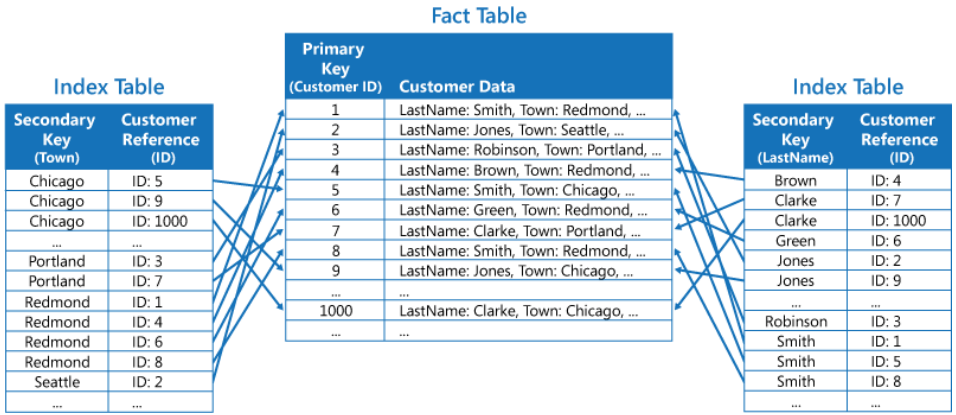
```
CREATE TABLE projects (
        project_id int(6) PRIMARY KEY,
        dept_id int(4),
        FOREIGN KEY (dept_id) REFERENCES departments (department_id) ON UPDATE CASCADE
                                                                     ON DELETE CASCADE
);

INSERT INTO departments (department_id, department_name) VALUES (1234, 'New department');
INSERT INTO projects VALUES (987, 1234);

UPDATE departments SET department_id = 1235 WHERE department_id = 1234;
// Check projects table. Reference updated

DELETE FROM departments WHERE department_id = 1235;
// Check projects table. Project deleted
```

- Indices
  - An **index** contains an entry for each value that appears in the indexed column(s) and provides direct, fast access to rows (B-tree default)



Syntax: **CREATE INDEX** *myindexfooname* **ON** *table*(column);
Example: **CREATE INDEX** *emp_job_ix* **ON** *employees*(job_id);
See: Optimizing Queries with EXPLAIN and indices

- Functions and procedures
  - **Functions**: a function is invoked within a SQL expression and returns a single value directly to the caller to be used in the calling SQL expression. **Read only**!
  - **Procedures**: it is invoked with a CALL statement to perform an operation such as modifying a table. A procedure does not return a value.
  - **Differences**:
    - Procedures can be called by themselves, while functions are called as **part of** a SQL expression. Functions return values to the caller.
    - We cannot invoke a function with a CALL statement.
    - We cannot invoke a procedure within an expression.
    - Procedure parameters can be defined as input-only, output-only, or for both input and output. Functions have only input parameters.

- Functions
  - Accept optional input parameter(s) and return some data
  - A RETURN statement **must** contain an expression or variable

```
DELIMITER //          -- change delimiter to //

CREATE FUNCTION function_name
        ( [parameter datatype [, parameter datatype …]] )
RETURNS return_datatype
BEGIN
        -- Declarations section (local variables and cursors)
        -- Executable section (logic of the function)
        RETURN result;
END//

DELIMITER ;          -- change delimiter back to ;
```

- Functions with no input parameters

```
DELIMITER //
CREATE FUNCTION getNameOfPresident() RETURNS VARCHAR(255)
BEGIN
    DECLARE v_name varchar(255);

    SELECT CONCAT(first_name, ' ', last_name) INTO v_name
    FROM employees
    WHERE employee_id = '100';

    RETURN v_name;
END//
DELIMITER ;
```

RETURN TYPE

VARIABLE DECLARATION

ASSIGN VALUE TO VARIABLE

RETURN RESULT

How to call a function?

```
getNameOfPresident();                  -- nope, this is a function
SELECT * FROM getNameOfPresident();    -- nope, this is a function
SELECT getNameOfPresident() FROM dual; -- yesss
select getNameOfPresident() from employees;  -- called once PER ROW for each employee
```

- Functions with input parameters

```
DELIMITER //
CREATE FUNCTION getNameOfEmployee(p_emp_id  INT) RETURNS VARCHAR(255)
BEGIN
        DECLARE v_name varchar(255);
        DECLARE v_text    varchar(255);

        SELECT CONCAT(first_name, ' ', last_name) INTO v_name
        FROM employees
        WHERE employee_id = p_emp_id;

        SET v_text = CONCAT ('Name of employee is ', v_name);
        RETURN v_text;
END//
DELIMITER ;
```

INPUT PARAMETERS

RETURN TYPE

VARIABLE DECLARATION

ASSIGN VALUE TO VARIABLE

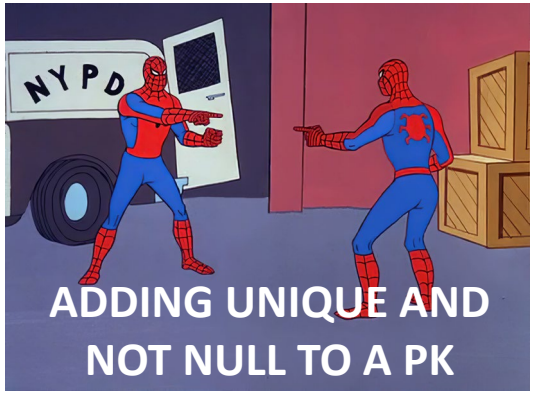
SET VALUE TO VARIABLE

RETURN RESULT

How to call a function?

SELECT getNameOfEmployee(employee_id) FROM employees; -- yesss

- Exercise
  - Write a function to calculate the salary difference between an employee and their manager. The employee ID is provided as input argument to the function.



- Procedures
  - Procedures may modify tables, and comprise many input / output parameters
  - Procedures do **not** contain a return statement

```
DELIMITER //

CREATE PROCEDURE procedure_name
        ( [ [IN | OUT | INOUT] parameter datatype … ] )
BEGIN
        -- Declarations section (local variables and cursors)
        -- Executable section (logic of the procedure)
END//

DELIMITER ;
```

- Procedures
  - Example: get the salary of a given employee

```
DELIMITER //

CREATE PROCEDURE getSalary (IN p_employee_id INT, OUT p_salary VARCHAR(255))
BEGIN
   SELECT concat('USD ', format(salary,2)) INTO p_salary
   FROM employees WHERE employee_id = p_employee_id;
END//

DELIMITER ;

CALL getSalary (100, @salary);        -- yes, out parameter is stored in temp variable @salary
SELECT @salary;                       -- show value of the variable
SELECT getSalary(employee_id, @salary), salary FROM employees;   -- nope, this is not a function
```

- Procedures
  - Example: increase the salary of a given employee

```
DELIMITER //

CREATE PROCEDURE IncreaseSalary ( IN p_employee_id INT, IN p_increment_pct FLOAT )
BEGIN
    UPDATE employees SET salary = salary * (1 + p_increment_pct)
    WHERE employee_id = p_employee_id;
END//

DELIMITER ;
```

```
CALL IncreaseSalary (100, 0.1);
```

- Procedures
  - Example: increase and return the salary of a given employee

```
DELIMITER //

CREATE PROCEDURE IncreaseSalaryReturn  ( IN p_employee_id INT,
                                         IN p_increment_pct FLOAT,
                                         OUT p_new_salary DECIMAL(8,2))
BEGIN
   SELECT salary INTO p_new_salary FROM employees WHERE employee_id = p_employee_id;

   SET p_new_salary  = p_new_salary  * (1 + p_increment_pct);

   UPDATE employees SET salary = p_new_salary WHERE employee_id = p_employee_id;
END//

DELIMITER ;
```

```
CALL IncreaseSalaryReturn (100, 0.1, @newsal);
SELECT @newsal AS newsalary;
```

- Conditional statements

```
CREATE PROCEDURE getEmployeeLevel( IN  p_employee_id INT, OUT p_level VARCHAR(20))
BEGIN
   DECLARE v_salary DECIMAL(8,2);

   SELECT salary INTO v_salary
   FROM employees
   WHERE employee_id = p_employee_id;

   IF v_salary > 20000 THEN
      SET p_level = 'PLATINUM';
   ELSEIF v_salary > 10000 THEN
      SET p_level = 'GOLD';
   ELSE
      SET p_level = 'SILVER';
   END IF;
END//
```

```
IF condition THEN
  -- IF SECTION
ELSEIF condition THEN
  -- ELSEIF SECTION
ELSE
  -- OTHERWISE
END IF;
```

- Loops

```
WHILE condition DO
  -- DO STUFF
END WHILE;
```

```
myloop: REPEAT
  -- DO STUFF
UNTIL condition
END REPEAT myloop;
```

```
myloop: LOOP
  IF condition THEN
     -- DO STUFF
    ITERATE myloop;
  END IF;
  LEAVE myloop;
END LOOP myloop;
```

```
CREATE PROCEDURE countDaysHired (IN p_employee_id INT,
BEGIN                            OUT p_days INT)
   DECLARE v_hire_date DATE;
   SET p_days = 0;

   SELECT hire_date INTO v_hire_date FROM employees WHERE employee_id = p_employee_id;

   WHILE v_hire_date < CURDATE() DO
     SET p_days = p_days + 1;
     SET v_hire_date = DATE_ADD(v_hire_date, INTERVAL 1 DAY);      -- not efficient, just educational to show a loop
   END WHILE;
END//
```

- Cursors
  - A cursor allows to iterate rows returned by a query and process each row

```
CREATE PROCEDURE getEmployeeNames (IN p_department_id INT)
BEGIN
        DECLARE done INT DEFAULT FALSE;
        DECLARE cur CURSOR FOR SELECT first_name FROM employees WHERE department_id = p_department_id;
        DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

          OPEN cur;
          myloop: LOOP
            FETCH cur INTO v_employee_name;
            IF done THEN
                LEAVE myloop;
            END IF;
            -- DO STUFF
          END LOOP;
          CLOSE cur;
END//
```

- Cursors

```
CREATE PROCEDURE sumSalaries ()
BEGIN
        …
        DECLARE done INT DEFAULT FALSE;
        DECLARE cur CURSOR FOR SELECT salary FROM employees;
        DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;
          OPEN cur;
          myloop: LOOP
            FETCH cur INTO v_salary;
            IF done THEN
                  LEAVE myloop;
            END IF;
                  IF v_salary > 10000 THEN
                        SET v_salary_high_sum = v_salary_high_sum + v_salary;
                  ELSE
                        SET v_salary_low_sum = v_salary_low_sum + v_salary;
                  END IF;
          END LOOP;
          CLOSE cur;
          SELECT CONCAT('Sum salaries >= $10k = ' , v_salary_high_sum, ' Sum salaries < $10k = ', v_salary_low_sum);
END//
```

- Temporary tables

```
CREATE PROCEDURE tempTableSumSalaries ()
BEGIN
  …
  CREATE TEMPORARY TABLE mytemptable ( higher_sal float, lower_sal  float );
  INSERT INTO mytemptable VALUES (0, 0);
  OPEN cur;
  myloop: LOOP
    FETCH cur INTO v_salary;
    …
        IF v_salary > 10000 THEN
            UPDATE mytemptable SET higher_sal = higher_sal + v_salary;
        ELSE
            UPDATE mytemptable SET lower_sal = lower_sal + v_salary;
        END IF;
  END LOOP;
  CLOSE cur;
  SELECT higher_sal, lower_sal INTO v_salary_high_sum, v_salary_low_sum FROM mytemptable;
  SELECT CONCAT('Sum salaries >= $10k = ' , v_salary_high_sum, ' Sum salaries < $10k = ', v_salary_low_sum);
  DROP TEMPORARY TABLE mytemptable;
END//
```

- Exercises **These exercises should be included in the second SQL homework.**
  - Create a function to return the manager's full name for an employee whose employee_id is provided as input parameter.
  - Create a function called format_phone. It will format the input argument 123.456.7890 so that it looks like a U.S. phone number (123) 456-7890.
  - Create a function to return the median salary for a department_id provided as input parameter.
  - Create a procedure to increase (increase_pct as parameter) the salary of the manager whose subordinate employee_id is provided as input parameter.
  - Create a procedure to create a table with the department name, the department's manager full name and the number of employees working for that department.
  - Create a procedure to increase 10% the salary of all subordinates in a department, do it as many times as necessary, until the average salary difference between managers and their subordinates in the department is smaller than 5%.

- 2020 test: **functions**

- Create a function named **employeesCountry** to calculate the number of employees working in a given country (including 0). The country_id is provided as the input parameter of the function.

- Write a query (calling the function created before) to show for every country how many employees work there.

- 2020 test: **procedures**

- Create a procedure named **INCREASE_SALARY_SUPERVISORS**. The procedure will increase the salary of each employee as follows:
  - 10% if the employee supervises one employee.
  - 15% if the employee supervises two employees.
  - 20% if the employee supervises three or more employees.
- The procedure will return an output parameter containing the sum of all of the salary increases.
- Execute the procedure print the output value.

CMSC 508
Databases

Advanced SQL (I)

VCU
College of Engineering

Dr. Alberto Cano
Associate Professor
Department of Computer Science

Chapter 3 from Database System Concepts, 7th Ed. by Silberschatz, Korth, Sudarshan, 2019
Chapter 5 from Database Management Systems, 3rd Ed. by Ramakrishnan, Gehrke, 2003