CMSC 508
Databases

Advanced SQL (III)

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

- Views
  - In some cases, it is not desirable for all users to see the entire logical model (all the actual tables and contents in the database)
  - Consider a person who needs to access an employee name and department, but not the salary. This person should see a partial view of the data described in SQL as

```
SELECT employee_id, last_name, department_id
FROM employees;
```

  - A view provides a mechanism to hide certain data (columns) from the view of certain users
  - Any relation that is not of the conceptual model but is made visible to a user as a "virtual relation" is called a **view**

- Views
  - A view is defined using the **create view** statement

    CREATE VIEW *viewName* [(*columns*)] **AS** < query expression >

    where <query expression> is any legal SQL expression and [(columns)] will name the respective output columns of the view. The view name is represented by *viewName*
  - Once a view is defined, the view name can be used to refer to the virtual relation that the view generates
  - Defining a view is **not** the same as creating a new relation by evaluating the query expression

- Views examples

```
CREATE VIEW employeesData (employee_id, last_name, department_name) AS
SELECT employee_id, last_name, department_name
FROM employees, departments
WHERE employees.department_id = departments.department_id;
```

```
CREATE VIEW employeesLocation (employee_id, last_name, city) AS
SELECT employee_id, last_name, city
FROM employees JOIN departments USING (department_id)
               JOIN locations USING (location_id);
```

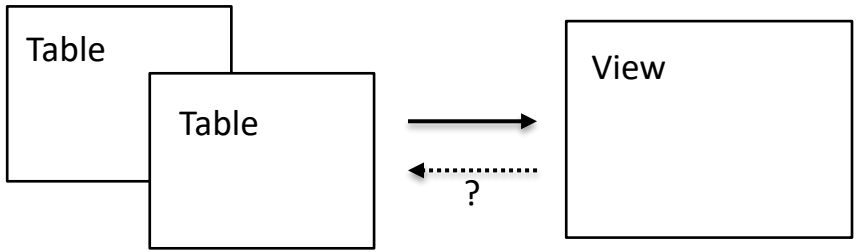
- Views using other views

```
CREATE VIEW employeesDataLocation (last_name, department_name, city) AS
SELECT employeesData.last_name, department_name, city
FROM employeesData JOIN employeesLocation USING (employee_id);
```

- Views filtering

```
CREATE VIEW highSalaryEmployees (last_name, department_name, salary) AS
SELECT e.last_name, d.department_name, e.salary
FROM employees e JOIN departments d
ON e.department_id = d.department_id
WHERE e.salary >
      (SELECT AVG(salary) FROM employees c
      WHERE e.department_id = c.department_id);
```

- Updatable and Insertable Views
  - Some views are updatable and references to them can be used to specify tables to be updated in DML statements. That is, you can use them in statements such as **UPDATE**, **DELETE**, or **INSERT** to update the contents of the underlying table.
  - For a view to be updatable, there must be a **one-to-one relationship** between the rows in the view and the rows in the underlying table.
  - Otherwise, the mapping cannot be automatically performed by the DBMS (e.g. when aggregate functions, GROUP BY, etc that "destroy" the traceability)



- Updatable and Insertable Views

#	Name	Type	Collation	Attributes	Null	Default
1	region_id	int(11)			No	*None*
2	region_name	varchar(25)	latin1_swedish_ci		Yes	*NULL*

CREATE VIEW *regionsView* **AS**
SELECT *region_id, region_name* **FROM** *regions*;

INSERT INTO *regionsView* **VALUES** (99,'Antarctica');

UPDATE *regionsView*
SET *region_name* = 'Freezingland' **WHERE** *region_id* = 99;

DELETE FROM *regionsView* **WHERE** *region_id* = 99;

Yes, the actions on the view can be mapped to actions on the table

DROP VIEW *regionsView;*

- Updatable and Insertable Views
  - Specifically, a view is **not** updatable if it contains **any** of the following:
    - Aggregate functions
    - DISTINCT, GROUP BY, HAVING, UNION
    - Subquery in the select list
    - Reference to nonupdatable view in the FROM clause
    - Subquery in the WHERE clause that refers to a table in the FROM clause
    - Refers only to literal values
    - Multiple references to any column of a base table

- Updatable and Insertable Views
  - Example: view is **not** updatable

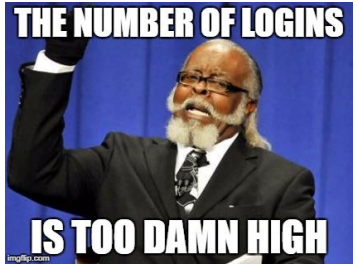
CREATE VIEW *employeesDataLocation* (*last_name, department_name, city*) **AS**
SELECT *employeesData.last_name, department_name, city*
FROM *employeesData* **JOIN** *employeesLocation*
ON *employeesData.employee_id* = *employeesLocation.employee_id*;

INSERT INTO *employeesDataLocation*
VALUES ('John', 'Marketing', 'Seattle');

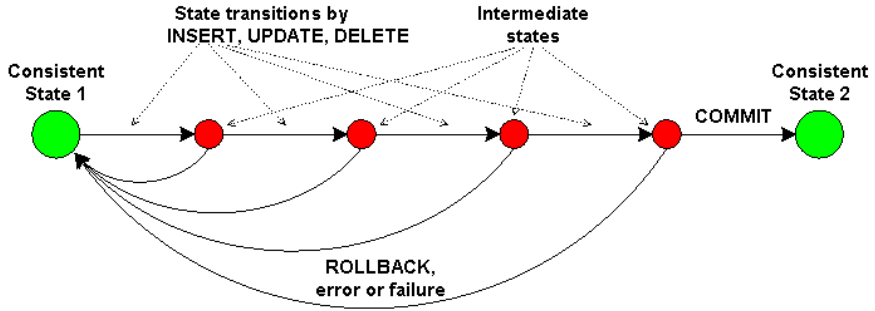
```
#1394 - Can not insert into join view 'employeesDataLocation' without fields list
```

What's the employeeID, deparmentID, salary, etc?

- Authorization
  - In some cases, it is not desirable for all users to see the entire logical model (all the actual relations and contents in the database)
  - Create users whose only privileges are to read the views pre-defined for them
  - Global vs local privileges for each user / database / table / action
  - This way you can hide any information to any user / 3-rd party API
  - Any modifications in the internal design of the database will simply need a modification of the internal view definition, while 3-rd party APIs will continue working unaware that a change happened on the underlying tables



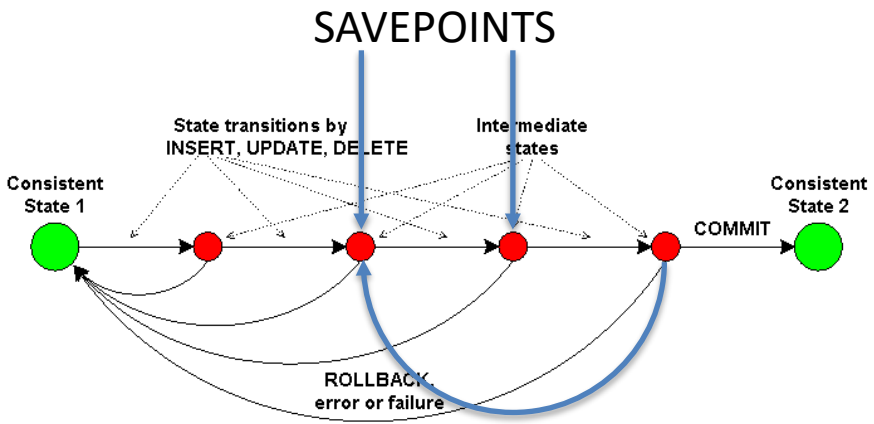
- Transactions
  - A transaction is a sequence of SQL statements treated as a single unit. ACID.
  - **START TRANSACTION**: to declare the beginning of a transaction
  - **COMMIT**: To commit the current transaction and make its changes permanent
  - **ROLLBACK**: To roll back the current transaction and cancel its changes
  - **SET autocommit** = ON | OFF to enable/disable autocommits
  - Additional slides on Canvas with in-depth description



- Transactions

```
SET autocommit = OFF;
SHOW VARIABLES WHERE Variable_name='autocommit';

START TRANSACTION;

SELECT @employee_id:=employee_id
FROM employees
WHERE salary = (SELECT MIN(salary) FROM employees);

SELECT * FROM employees;

DELETE FROM employees
WHERE employee_id = @employee_id;

SELECT * FROM employees;

ROLLBACK;  -- use to undo the operations of the transaction
COMMIT;    -- use to make permanent the changes of the transaction
```

- Transactions
  - A transaction is a sequence of SQL statements treated as a single unit. ACID.
  - **SAVEPOINT**: sets a named transaction savepoint with a name of the identifier
  - **ROLLBACK TO SAVEPOINT**: statement rolls back a transaction to the named savepoint without terminating the transaction
  - **RELEASE SAVEPOINT**: to release a savepoint named earlier



- Transactions

```
SET autocommit = OFF;
SHOW VARIABLES WHERE Variable_name='autocommit';

START TRANSACTION;

SAVEPOINT p1; -- creates a savepoint

DELETE FROM employees WHERE employee_id = 132;

SAVEPOINT p2; -- creates a savepoint

DELETE FROM employees WHERE employee_id = 128;

ROLLBACK TO SAVEPOINT p2;    -- undo changes and restore savepoint p2

ROLLBACK TO SAVEPOINT p1;    -- undo changes and restore savepoint p1
```

- Table locking and critical sections
  - MySQL enables sessions to acquire table locks to prevent other sessions from modifying tables during periods requiring exclusive access to them.
  - A session can acquire or release locks only for itself. One session cannot acquire locks for another session or release locks held by another session.
  - **LOCK TABLES** acquires table locks for the current client session: READ | WRITE
  - **UNLOCK TABLES** releases any table locks held by the current session
  - Check out this very well written article

- Exercises **These exercises should be included in the second SQL homework.**

- Create a function to compute the average of the salaries for a given department_id defined as input parameter of the function. Then, create a view named department_statistics as the result of a query collecting for every department: the department name, name of the department manager (in the form “F. LastName” where F. is the first letter of the first name), the number of employees working for the department, the lowest and highest salary of its employees, and the average of the salaries (as a result of the call of the function you created first). Include in the view departments not having any employee and display 0 rather than NULL when necessary.
- Create a view to find for each department the largest salary difference between any two employees within the department.
- Create a view to find the number of departments in each region, including regions with 0 departments (show 0) and departments with no regions assigned (show ‘NO REGION’).
- Create a procedure protected with a transaction to steal a percentage of the salary of the employees and transfer the money to the president’s salary. If the president’s new salary is bigger than $100k undo the changes.

CMSC 508
Databases

Advanced SQL (III)

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