

More SQL: Complex Queries, Triggers, Views, and Schema Modification

Part 2

Specifying Constraints as
Assertions and Actions as Triggers

Specifying General Constraints as Assertions in SQL

- ```
CREATE ASSERTION SALARY_CONSTRAINT
CHECK (NOT EXISTS (SELECT *
 FROM EMPLOYEE E, EMPLOYEE M,
 DEPARTMENT D
 WHERE E.Salary>M.Salary AND
 E.Dno=D.Dnumber AND
 D.Mgr_ssn=M.Ssn));
```

# Introduction to Triggers in SQL

- **R5:**

```
CREATE TRIGGER SALARY_VIOLATION
BEFORE INSERT OR UPDATE OF SALARY, SUPER_SSN
ON EMPLOYEE
FOR EACH ROW
WHEN (NEW.SALARY > (SELECT SALARY
 FROM EMPLOYEE
 WHERE SSN = NEW.SUPER_SSN))
INFORM_SUPERVISOR(NEW.Super_ssn, NEW.Ssn);
```

# Views (Virtual Tables) in SQL

# Concept of a View in SQL

- A **view** in SQL terminology is a single table that is derived from other tables.
- These other tables can be *base tables* or previously defined views.
- A view does not necessarily exist in physical form; it is considered to be a **virtual table**, in contrast to **base tables**, whose tuples are always physically stored in the database.
- This limits the possible update operations that can be applied to views, but it does not provide any limitations on querying a view.

# Specification of Views in SQL

- **V1:**

```
CREATE VIEW WORKS_ON1 AS
SELECT Fname, Lname, Pname, Hours
FROM EMPLOYEE, PROJECT, WORKS_ON
WHERE Ssn=Essn AND Pno=Pnumber;
```
- **V2:**

```
CREATE VIEW DEPT_INFO (Dept_name, No_of_emps,
 Total_sal) AS
SELECT Dname, COUNT(*), SUM(Salary)
FROM DEPARTMENT, EMPLOYEE
WHERE Dnumber=Dno
GROUP BY Dname;
```

- QV1:       SELECT Fname, Lname  
              FROM     WORKS\_ON1  
              WHERE    Pname='ProductX';



- A view is supposed to be *always up-to-date*; if we modify the tuples in the base tables on which the view is defined, the view must automatically reflect these changes.
- Hence, the view is not realized or materialized at the time of *view definition* but rather at the time when we *specify a query* on the view.
- It is the responsibility of the DBMS and not the user to make sure that the view is kept up-to-date.

- If we do not need a view any more, we can use the **DROP VIEW** command to dispose of it.
- V1A:        `DROP VIEW WORKS_ON1;`

# View Implementation, View Update, and Inline Views

- The problem of efficiently implementing a view for querying is complex.
- Two main approaches have been suggested.
- One strategy, called **query modification**, involves modifying or transforming the view query (submitted by the user) into a query on the underlying base tables.

- QV1:       SELECT Fname, Lname  
              FROM       WORKS\_ON1  
              WHERE     Pname='ProductX';
- SELECT Fname, Lname  
      FROM     EMPLOYEE, PROJECT, WORKS\_ON  
      WHERE    Ssn=Essn AND Pno=Pnumber AND  
              Pname= 'ProductX';

- The disadvantage of this approach is that it is inefficient for views defined via complex queries that are time-consuming to execute, especially if multiple queries are going to be applied to the same view within a short period of time.

- The second strategy, called **view materialization**, involves physically creating a temporary view table when the view is first queried and keeping that table on the assumption that other queries on the view will follow.
- In this case, an efficient strategy for automatically updating the view table when the base tables are updated must be developed in order to keep the view up-to-date.
- Techniques using the concept of **incremental update** have been developed for this purpose, where the DBMS can determine what new tuples must be inserted, deleted, or modified in a *materialized view table* when a database update is applied *to one of the defining base tables*.
- The view is generally kept as a materialized (physically stored) table as long as it is being queried.
- If the view is not queried for a certain period of time, the system may then automatically remove the physical table and recompute it from scratch when future queries reference the view.

- Updating of views is complicated and can be ambiguous.
- In general, an update on a view defined on a *single table* without any *aggregate functions* can be mapped to an update on the underlying base table under certain conditions.
- For a view involving joins, an update operation may be mapped to update operations on the underlying base relations in *multiple ways*.
- Hence, it is often not possible for the DBMS to determine which of the updates is intended.

- **V1:** CREATE VIEW WORKS\_ON1 AS  
SELECT Fname, Lname, Pname, Hours  
FROM EMPLOYEE, PROJECT, WORKS\_ON  
WHERE Ssn=Essn AND Pno=Pnumber;
- **UV1:** UPDATE WORKS\_ON1  
SET Pname='ProductY'  
WHERE Lname='Smith' AND  
Fname='John' AND  
Pname='ProductX';



- (a): UPDATE WORKS ON  
SET Pno=(SELECT Pnumber  
FROM PROJECT  
WHERE Pname='ProductY')  
WHERE Essn IN (SELECT Ssn  
FROM EMPLOYEE  
WHERE Lname='Smith' AND  
Fname='John')  
  
AND  
Pno = (SELECT Pnumber  
FROM PROJECT  
WHERE Pname='ProductX') ;
- (b): UPDATE PROJECT  
SET Pname = 'ProductY'  
WHERE Pname = 'ProductX';

- Some view updates may not make much sense; for example, modifying the `Total_sal` attribute of the `DEPT_INFO` view does not make sense because `Total_sal` is defined to be the sum of the individual employee salaries.
- UV2:  

```
UPDATE DEPT_INFO
SET Total_sal=100000
WHERE Dname='Research';
```

- Generally, a view update is feasible when only *one possible update* on the base relations can accomplish the desired update effect on the view.
- Whenever an update on the view can be mapped to *more than one update* on the underlying base relations, we must have a certain procedure for choosing one of the possible updates as the most likely one.

- It is also possible to define a view table in the **FROM clause** of an SQL query.
- This is known as an **in-line view**.

# Schema Change Statements in SQL

# The DROP Command

- If a whole schema is no longer needed, the `DROP SCHEMA` command can be used.
- There are two drop behavior options: `CASCADE` and `RESTRICT`.
- To remove the `COMPANY` database schema and all its tables, domains, and other elements, the `CASCADE` option is used as follows:

```
DROP SCHEMA COMPANY CASCADE
```

- If the `RESTRICT` option is chosen in place of `CASCADE`, the schema is dropped only if it has no elements in it; otherwise, the `DROP` command will not be executed.

- If we no longer wish to keep track of dependents of employees in the COMPANY database, we can get rid of the DEPENDENT relation by issuing the following command:

```
DROP TABLE DEPENDENT CASCADE
```

- If the RESTRICT option is chosen instead of CASCADE, a table is dropped only if it is not referenced in any constraints or views or by any other elements.
- With the CASCADE option, all such constraints, views, and other elements that reference the table being dropped are also dropped automatically from the schema, along with the table itself.

- Notice that the `DROP TABLE` command not only deletes all the records in the table if successful, but also removes the table definition from the catalog.
- If it is desired to delete only the records but to leave the table definition for future use, then the `DELETE` command should be used instead of `DROP TABLE`.



# The ALTER Command

- To add an attribute for keeping track of jobs of employees to the `EMPLOYEE` base relation in the `COMPANY` schema, we can use the command

```
ALTER TABLE COMPANY.EMPLOYEE
ADD COLUMN Job VARCHAR(12) ;
```

- We must still enter a value for the new attribute `Job` for each individual `EMPLOYEE` tuple.
- This can be done either by specifying a default clause or by using the `UPDATE` command individually on each tuple.

- To drop a column, we must choose either `CASCADE` or `RESTRICT` for drop behavior.
- If `CASCADE` is chosen, all constraints and views that reference the column are dropped automatically from the schema, along with the column.
- If `RESTRICT` is chosen, the command is successful only if no views or constraints (or other schema elements) reference the column.
- ```
ALTER TABLE COMPANY.EMPLOYEE  
DROP COLUMN Address CASCADE;
```

- It is also possible to alter a column definition by dropping an existing default clause or by defining a new default clause.
- `ALTER TABLE COMPANY.DEPARTMENT ALTER COLUMN Mgr_ssn DROP DEFAULT;`
- `ALTER TABLE COMPANY.DEPARTMENT ALTER COLUMN Mgr_ssn SET DEFAULT '333445555';`

- One can also change the constraints specified on a table by adding or dropping a named constraint.
- To be dropped, a constraint must have been given a name when it was specified.

```
ALTER TABLE COMPANY.EMPLOYEE DROP CONSTRAINT  
EMPSUPERFK CASCADE;
```

- To add a new constraint to the relation, the `ADD` keyword in the `ALTER TABLE` statement can be used.