Views, assertions, and triggers

- **Views** are a mechanism for customizing the database; also used for creating temporary virtual tables
- **Assertions** provide a means to specify additional constraints
- **Triggers** are a special kind of assertions; they define actions to be taken when certain conditions occur

Reminder: basic Architecture of a Database System

[Diagram showing the architecture of a database system with levels and views.]
Views

- In some cases, it is not desirable for all users to see the entire logical model (i.e., all the actual relations stored in the database.)
- Consider a person who needs to know customers’ loan numbers but has no need to see the loan amounts. This person should see a relation described, in SQL, by
  
  \[
  \text{select customer_name, loan_number} \\
  \text{from customer c, borrower b} \\
  \text{where c.customer_id = b.customer_id}
  \]
- A view provides a mechanism to hide or restructure data for certain users.
- Any relation that is not in the database schema but is made visible to a user as a “virtual relation” is called a view.

Bank relational schema

- \( \text{branch} = (\text{branch_name, branch_city, assets}) \)
- \( \text{loan} = (\text{loan_number, branch_name, amount}) \)
- \( \text{account} = (\text{account_number, branch_name, balance}) \)
- \( \text{borrower} = (\text{customer_id, loan_number}) \)
- \( \text{depositor} = (\text{customer_id, account_number}) \)
- \( \text{customer} = (\text{customer_id, customer_name}) \)
View Definition

• A view is defined using the `create view` statement which has the form

```
create view V as <query expression>
```

where $V$ is the view name and $<query$ expression$>$ is any legal SQL query. A list of attribute names for $V$ is optional.

• Once a view is defined, the view name can be used in queries

• Only limited updates can be applied to the view (more later)

• View definition is not the same as creating a new relation by evaluating the query expression: the view contents is refreshed automatically when the database is updated

Examples

• A view consisting of bank branches and all their customers

```
create view `all_customers` as
    (select branch_name, customer_id
     from depositor d, account a
     where d.account_number = a.account_number)
union
    (select branch_name, customer_id
     from borrower b, loan l
     where b.loan_number = l.loan_number)
```

• Find all customers of the La Jolla branch

```
select customer_id
from `all_customers`
where branch_name = 'La Jolla'
```
Views Defined Using Other Views

• One view may be used in the expression defining another view.

• A view relation $V_1$ is said to depend directly on a view relation $V_2$ if $V_2$ is used in the expression defining $V_1$.

• A view relation $V_1$ is said to depend on view relation $V_2$ if either $V_1$ depends directly to $V_2$ or there is a path of dependencies from $V_1$ to $V_2$.

• A view relation $V$ is said to be recursive if it depends on itself $\rightarrow$ will discuss later…

<table>
<thead>
<tr>
<th>Record</th>
<th>ID</th>
<th>Qtr</th>
<th>Year</th>
<th>Class</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>S</td>
<td>2013</td>
<td>2013</td>
<td>100</td>
<td>B+</td>
</tr>
<tr>
<td>111</td>
<td>W</td>
<td>2014</td>
<td>2014</td>
<td>131</td>
<td>A-</td>
</tr>
<tr>
<td>111</td>
<td>F</td>
<td>2018</td>
<td>2018</td>
<td>132A</td>
<td>null</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>GPAS</th>
<th>ID</th>
<th>Acad-Year</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>2012/13</td>
<td>3.4</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ID</th>
<th>Acad-Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>2012/13</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Year</th>
<th>Qtr</th>
<th>Acad-Year</th>
</tr>
</thead>
<tbody>
<tr>
<td>2013</td>
<td>S</td>
<td>2012/13</td>
</tr>
<tr>
<td>2014</td>
<td>W</td>
<td>2013/14</td>
</tr>
<tr>
<td>2016</td>
<td>F</td>
<td>2016/17</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Grade</th>
<th>Points</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>4.0</td>
</tr>
<tr>
<td>A-</td>
<td>3.75</td>
</tr>
<tr>
<td>B+</td>
<td>3.25</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Record</th>
<th>ID</th>
<th>Qtr</th>
<th>Year</th>
<th>Class</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td>111</td>
<td>S</td>
<td>2013</td>
<td>2013</td>
<td>100</td>
<td>B+</td>
</tr>
<tr>
<td>111</td>
<td>W</td>
<td>2014</td>
<td>2014</td>
<td>131</td>
<td>A-</td>
</tr>
<tr>
<td>111</td>
<td>F</td>
<td>2018</td>
<td>2018</td>
<td>132A</td>
<td>null</td>
</tr>
<tr>
<td>Drop-stats</td>
<td>ID</td>
<td>Acad-Year Drops</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>------------</td>
<td>------</td>
<td>-----------------</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>111</td>
<td>2012/13</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td>111</td>
<td>2016/17</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Need triggers:
on deletion increase Drops

<table>
<thead>
<tr>
<th>Record</th>
<th>ID</th>
<th>Qtr</th>
<th>Year</th>
<th>Class</th>
<th>Grade</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>111</td>
<td>S</td>
<td>2013</td>
<td>100</td>
<td>B+</td>
</tr>
<tr>
<td></td>
<td>111</td>
<td>W</td>
<td>2014</td>
<td>131</td>
<td>A-</td>
</tr>
<tr>
<td></td>
<td>111</td>
<td>F</td>
<td>2018</td>
<td>132A</td>
<td>null</td>
</tr>
</tbody>
</table>

Views can simplify complex queries

Example

find actors playing in every movie by “Berto”:

SELECT Actor FROM Movie
WHERE Actor NOT IN

(SELECT m1.Actor
FROM Movie m1, Movie m2,
WHERE m2.Director="Berto"
AND m1.Actor NOT IN
(SELECT Actor
FROM Movie
WHERE Title=m2.Title))

The shaded query finds actors NOT playing in some movie by “Berto”
Same query using views

```
CREATE VIEW Berto-Movies AS
SELECT title FROM Movie WHERE director = "Bertolucci";
```

```
CREATE VIEW Not-All-Berto AS
SELECT m.actor FROM Movies m, Berto-Movies
WHERE  Berto-Movies.title NOT IN
    (SELECT title FROM Movies
     WHERE actor = m.actor);
```

```
SELECT actor FROM Movies WHERE actor NOT IN
    (SELECT * FROM Not-All-Berto)
```

Another syntax: the `with` clause

```
WITH Berto-Movies AS
SELECT title FROM Movie WHERE director = "Bertoucci"

WITH Not-All-Berto AS
SELECT m.actor FROM Movies m, Berto-Movies
WHERE  Berto-Movies.title NOT IN
    (SELECT title FROM Movies
     WHERE actor = m.actor)

SELECT actor FROM Movies WHERE actor NOT IN
    (SELECT * FROM Not-All-Berto) ;
```

Note: Berto-Movies and Not-All-Berto are temporary tables, not views
Efficient View Implementation

- **Materialized views:**
  physically create and maintain a view table
  - assumption: other queries on the view will follow
  - concerns: maintaining correspondence between the base table and the view when the base table is updated
  - strategy: incremental update

- **Virtual views:**
  never physically created
  answer queries on the view by reformulating it as a query on the underlying base tables (by replacing the views by their definitions)
  - disadvantage: inefficient for views defined via complex queries (especially if additional queries are to be applied to the view within a short time period)
  - advantage: no need to maintain correspondence with base tables
Example of view unfolding:

CREATE VIEW Berto-Movies AS
SELECT title FROM Movie WHERE director = “Berto”;

SELECT theater FROM schedule WHERE title IN
(SELECT * FROM Berto-Movies)

SELECT theater FROM schedule WHERE title IN
(SELECT title FROM Movie WHERE director = “Berto”)

Answer
Q (View)
View
Q(V(DB))
V(DB)
DB
Another example of view unfolding

Database:

<table>
<thead>
<tr>
<th>Patient</th>
<th>pid</th>
<th>hospital</th>
<th>docid</th>
<th>Doctor</th>
<th>docid</th>
<th>docname</th>
</tr>
</thead>
</table>

View (Scripps doctors):

create view ScrippsDoc as
select d1.* from Doctor d1, Patient p1
where p1.hospital = 'Scripps' and p1.docid = d1.docid

View (Scripps patients):

create view ScrippsPatient as
select p2.* from Patient p2
where hospital = 'Scripps'

Scripps query (using views):

select p.pid, d.docname
from ScrippsPatient p, ScrippsDoc d
where p.docid = d.docid

Query on database obtained by view unfolding

query using view

select p.pid, d.docname
from ScrippsPatient p, ScrippsDoc d
where p.docid = d.docid

create view ScrippsDoc as
select d1.* from Doctor d1, Patient p1
where p1.hospital = 'Scripps' and p1.docid = d1.docid

create view ScrippsPatient as
select p2.* from Patient p2
where p2.hospital = 'Scripps'

result of view unfolding

select p.pid, d.docname
from Patient p, Doctor d, Patient p1
where p.docid = d.docid and p.hospital = 'Scripps'
and p1.hospital = 'Scripps' and p1.docid = d.docid
View updates: example

create view Berto-titles as
select title from movie where director = 'Bertolucci'

- deleting a title T in view → delete all tuples with title T from movie
- insert a title T in view → insert <T, ‘Bertolucci’, NULL> in movie
- update “Sky” to “Sheltering Sky” in view
  → update movie
  set title = ‘Sheltering Sky’
  where director = ‘Bertolucci’ and title = ‘Sky’

View updates: example

create view Same as
select t.theater, s.theater
from schedule t, schedule s
where t.title = s.title

- Suppose I insert <Ken, Hillcrest> in Same
  Problem: cannot be mapped to an update of movie because the common title is unknown
- Similar problem for deletes and updates
- Such view updates are prohibited
View Updates (cont)

• Update on views without aggregates, nesting, group-by, or tuple aliases, defined on a single base table, maps naturally to an update of the underlying base table
• For other views, mapping updates to base tables is not always possible
• Most SQL implementations allow updates only on simple views (without aggregates, nesting, group-by or tuple aliases) defined on a single base table

Assertions

• An assertion defines a constraint the database must satisfy
• An assertion in SQL takes the form
  
  `create assertion <assertion-name> check <predicate>`

• When an assertion is made, the system tests it for validity, and tests it again on every update that may violate the assertion
  
  Testing may introduce a significant amount of overhead; hence assertions should be used with great care.

• Asserting
  
  for all $X$, $P(X)$

  is achieved in a round-about fashion using
  
  not exists $X$ such that not $P(X)$
Using General Assertions

- Specify a query that violates the condition; include inside a `NOT EXISTS` clause
- Query result must be empty
  - if the query result is not empty, the assertion has been violated

Assertion Example

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

```sql
create assertion balance_constraint check (not exists
  (select *
    from loan
    where not exists
      (select *
        from borrower, depositor, account
        where loan.loan_number = borrower.loan_number
        and borrower.customer_id = depositor.customer_id
        and depositor.account_number = account.account_number
        and account.balance >= 1000.00)))
```
Bank schema

- branch = (branch_name, branch_city, assets)
- loan = (loan_number, branch_name, amount)
- account = (account_number, branch_name, balance)
- borrower = (customer_id, loan_number)
- depositor = (customer_id, account_number)
- customer = (customer_id, customer_name)

Assertion Example

- The sum of all loan amounts for each branch must be less than the sum of all account balances at the branch.

```sql
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(amount)
      from account
      where account.branch_name =
        branch.branch_name )))
```
Assertions: Another Example

• “The salary of an employee must not be greater than the salary of the manager of the department that the employee works for”

CREATE ASSERTION SALARY_CONSTRAINT
CHECK (NOT EXISTS
    (SELECT *
     FROM EMPLOYEE E, EMPLOYEE M, DEPARTMENT D
     WHERE E.SALARY > M.SALARY
     AND E.DNO=D.NUMBER
     AND D.MGRSSN=M.SSN))

SQL Triggers

• Objective: to monitor a database and take action when a condition occurs
• Triggers are expressed in a syntax similar to assertions and include the following:
  – event (an update operation)
  – condition
  – action (to be taken when the condition is satisfied)
SQL Triggers: An Example

- A trigger to compare an employee’s salary to his/her supervisor during insert or update operations:

```sql
CREATE TRIGGER INFORM_SUPERVISOR
BEFORE INSERT OR UPDATE OF
    SALARY, SUPERVISOR_SSN ON EMPLOYEE
FOR EACH ROW
WHEN (NEW.SALARY >
    (SELECT SALARY FROM EMPLOYEE
WHERE SSN=NEW.SUPERVISOR_SSN))
INSERT INTO INFORM_SUPERVISOR VALUES
    (NEW.SUPERVISOR_SSN, SSN);
```
Triggers (cont)

- Many variations in syntax, functionality
- Many triggering semantics possible:
  before/after event, immediate/deferred execution, etc.
- Behavior can be hard to anticipate
  sometimes results in non-terminating computations!
- Sub-area of databases: “Active databases”

A safe form of trigger: cascade

Enforces referential integrity

```sql
create table account
(account_number char(10),
branch_name   char(15),
balance       integer,
primary key   (account_number),
foreign key    (branch_name) references branch )
on delete cascade
```

Semantics: if a tuple deletion in branch causes a violation of referential integrity for some tuple t in account, the tuple t is also deleted