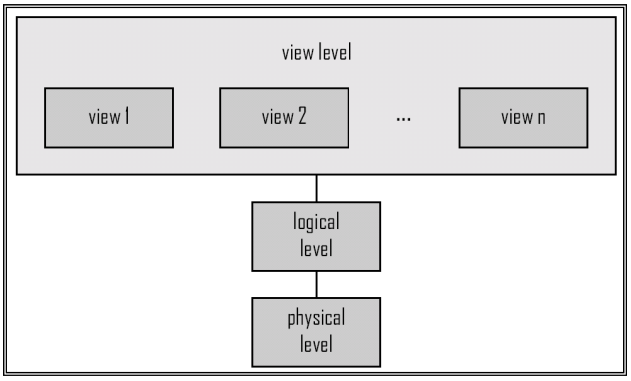
# Views, assertions, and triggers

- 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

1

Reminder: basic Architecture of a Database System




2

# 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

3

# 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

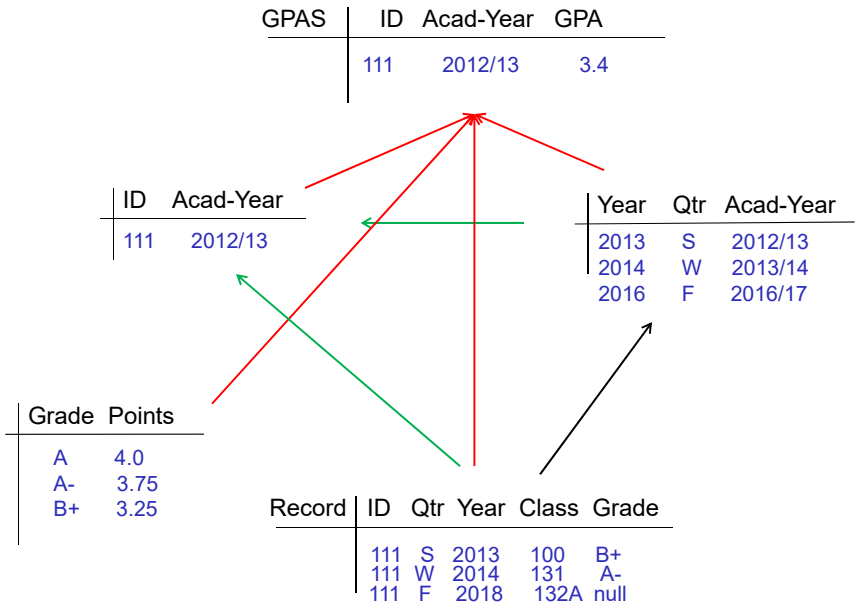
```
select customer_id
from all_customers
where branch_name = 'La Jolla'
```

4

# 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 → will discuss later…

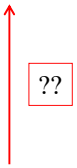
5




6

| Drop-stats | ID | Acad-Year | Drops |
|---|---|---|---|
| | 111 | 2012/13 | 2 |
| | 111 | 2016/17 | 3 |

Need triggers:
on deletion increase Drops




| Record | ID | Qtr | Year | Class | Grade |
|---|---|---|---|---|---|
| | 111 | S | 2013 | 100 | B+ |
| | 111 | W | 2014 | 131 | A- |
| | 111 | F | 2018 | 132A | null |

7

# Efficient View Implementation

- Materialized views:

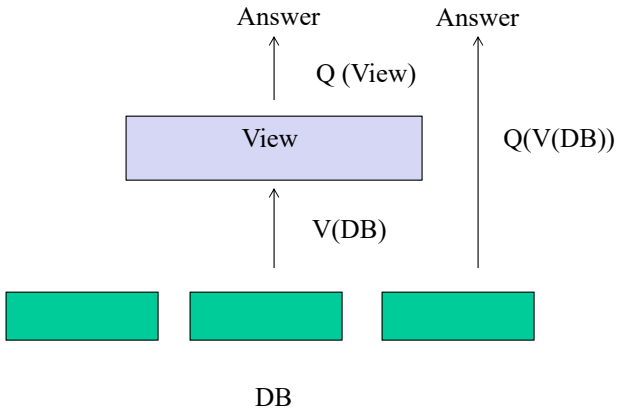
  physically create and maintain a view table
  - assumption: other queries on the view will follow
  - concerns: maintaining correspondence between the base table and the view when the base table is updated
  - strategy: incremental update

8

# Efficient View Implementation

- 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

9




10

## Example of view unfolding:

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

View

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

Query



```
SELECT theater FROM schedule WHERE title IN
        (SELECT title FROM Movie WHERE director = "Berto" )
```

11

## Another example of view unfolding

Database:

Patient | pid hospital docid

Doctor | docid  docname

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
```

12

## 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
```

view1

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

view2

```
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
```

13

## 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'

14

## View updates: example

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

*Same contains pairs of theaters showing the same 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

15

## 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

16

# 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 $P(X)$

17

# 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

18

## Assertion Example

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

**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)))

19

## 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)*

20

## Assertion Example

- The sum of all loan amounts for each branch must be less 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 (amount )
                           from account
                           where account.branch_name =
                                 branch.branch_name )))
```

21

## 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))
```

22

# 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)

23

Drop-stats

| ID | Acad-Year | Drops |
|---|---|---|
| 111 | 2012/13 | 2 |
| 111 | 2016/17 | 3 |

Trigger:
on deletion increase Drops

| Record | ID | Qtr | Year | Class | Grade |
|---|---|---|---|---|---|
| | 111 | S | 2013 | 100 | B+ |
| | 111 | W | 2014 | 131 | A- |
| | 111 | F | 2016 | 132A | null |

24

# SQL Triggers: An Example

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

```
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);
```

25

# 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"

26

# A safe form of trigger: cascade

Enforces referential integrity

**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

27