

SQL: Views

- Views
- Renaming View Attributes
- Using Views
- Updating Views
- Evaluating Views
- Materialized Views

❖ Views

A view is like a "virtual relation" defined via a query.

a virtual table that is generated from a query i.e. a set of tuples that gets generated on the fly when it is called.

View definition and removal:

```
CREATE VIEW ViewName AS Query
```

```
CREATE VIEW ViewName (AttributeNames) AS Query
```

```
DROP VIEW ViewName
```

Query may be any SQL query, involving: stored tables, other views

CREATE OR REPLACE replaces the *Query* associated with a view

Views (cont)

The stored tables used by a view are referred to as base tables.

Views are defined only after their base tables are defined.

A view is valid only as long as its underlying query is valid.

Dropping a view has no effect on the base tables.

Views are a convenient abstraction mechanism

- allow you to package and name complex queries
 - give you the "table that you wanted" to solve a more complex query
- without actually having to make said table as a concrete table in your schema!

Views (cont)

Example: defining/naming a complex query using a view:

```
CREATE VIEW
    CourseMarksAndAverages(course,term,student,mark,avg)
AS
SELECT s.code, termName(t.id), e.student, e.mark,
       avg(mark) OVER (PARTITION BY course)
FROM   CourseEnrolments e
       JOIN Courses c on c.id = e.course
       JOIN Subjects s on s.id = c.subject
       JOIN Terms t on t.id = c.term
;
```

which would make the following query easy to solve

```
SELECT course, term, student, mark
FROM   CourseMarksAndAverages
WHERE  mark < avg;
```

retrieve the marks for all students in a specific course session
whose marks were less than the average mark

Views (cont)

Example: An avid Carlton drinker might not be interested in other kinds of beer.

```
CREATE VIEW MyBeers AS
SELECT * FROM Beers WHERE brewer = 'Carlton';
```

which is used as

```
SELECT * FROM MyBeers;
```

name	brewer	style
Crown Lager	Carlton	Lager
Fosters Lager	Carlton	Lager
Invalid Stout	Carlton	Stout
Melbourne Bitter	Carlton	Lager
Victoria Bitter	Carlton	Lager

Views (cont)

A view might not use all attributes of the base relations.

Example: We don't really need the address of inner-city hotels.

```
CREATE VIEW InnerCityHotels AS
  SELECT name, license address would go here (if we wanted it)
  FROM Bars
  WHERE addr in ('The Rocks', 'Sydney');
```

```
SELECT * FROM InnerCityHotels;
```

name	license
Australia Hotel	123456
Lord Nelson	123888
Marble Bar	122123

Views (cont)

A view might use computed attribute values.

Example: Number of beers produced by each brewer.

```
CREATE VIEW BeersBrewed AS
  SELECT brewer, count(*) as nbeers
  FROM    beers GROUP BY brewer;
```

```
SELECT * FROM BeersBrewed;
```

brewer	nbeers
3 Ravens	1
Akasha	1
Alesmith	1
...	

❖ Renaming View Attributes

This can be achieved in two different ways:

```
CREATE VIEW InnerCityHotels AS
  SELECT name AS bar, license AS lic
  FROM Bars
  WHERE addr IN ('The Rocks', 'Sydney');
```

prefer this one, it more clearly shows what is being aliased to what

```
CREATE VIEW InnerCityHotels(bar, lic) AS
  SELECT name, license
  FROM Bars
  WHERE addr IN ('The Rocks', 'Sydney');
```

Both of the above produce the same view.

❖ Using Views

Views can be used in queries as if they were stored relations.

However, they differ from stored relations in two important respects:

- their "value" can change without being explicitly modified
(i.e. the result of a view may change whenever one of its base tables is updated)
- they may not be able to be explicitly modified (updated)
(only a certain simple kinds of views can be explicitly updated)

"Modifying a view" means changing the base tables via the view, e.g.

```
insert into MyBeers values ('Zero', 'Carlton', 'No-alcohol');
```

would update the **Beers** table (where MyBeers is a view)

COMP3311 20T3 ♦ SQL: Views ♦ [8/14]

It seems that you can insert into a view if the columns of the view provides enough information for the underlying base table. Note that you can only insert into views that have 1 base table, otherwise it wouldn't know which attributes are for which table, especially with shared attribute names like id or name.

For example, we can insert into mybeers because it has name,brewer,style and so it will actually insert into the beers table. We could omit style because it is not required. If the view omitted name however, then it wouldn't work as the underlying beers table requires style for inserts.

❖ Using Views (cont)

Example: of view changing when base table changes.

```
SELECT * FROM InnerCityHotels;
```

name	license
Australia Hotel	123456
Lord Nelson	123888
Marble Bar	122123

```
-- then the Lord Nelson goes broke
```

```
DELETE FROM Bars WHERE name = 'Lord Nelson';
```

```
-- no explicit update has been made to InnerCityHotels
```

```
SELECT * FROM InnerCityHotels;
```

name	license
Australia Hotel	123456
Marble Bar	122123

? gone, deleted

COMP3311 20T3 ♦ SQL: Views ♦ [9/14]

Because it is a view of something, if that something changes, then view changes

Imagine you are looking at something via a pair of binoculars (view). If someone changes the thing you are looking at (e.g. smashes it), you will see that change happen to your view.

Views are not a snapshot of a table at a particular point in time (static, analogous to a camera taking a photo instead of looking via binoculars), instead they are updated as the underlying table is updated (dynamic).

❖ Updating Views

Explicit updates are allowed on views satisfying the following:

- the view involves a single relation **R**
- the **WHERE** clause does not involve **R** in a subquery
- the **WHERE** clause only uses attributes from the **SELECT**

Attributes not in the view's **SELECT** will be set to **NULL** in the base relation after an insert into the view.

COMP3311 20T3 ♦ SQL: Views ♦ [10/14]

i.e. if we didn't select style in the mybeers view then it would be set to null in the beers table whenever we did an insert via mybeers

❖ Updating Views (cont)

Example: Our **InnerCityHotel** view is not updatable.

```
INSERT INTO InnerCityHotels  
VALUES ('Jackson' 's on George', '9876543');
```

creates a new tuple in the **Bars** relation:

(Jackson 's on George, **NULL**, 9876543)

but this new tuple **does not satisfy the view condition:**

addr IN ('The Rocks', 'Sydney')

so it does not appear if we select from the view.

❖ Evaluating Views

Two alternative ways of implementing views:

- re-writing rules (or macros)
 - when a view is used in a query, the query is re-written
 - after rewriting, becomes a query only on base relations
- explicit stored relations (called materialized views)
 - the view is stored as a real table in the database
 - updated appropriately when base tables are modified

The difference: underlying query evaluated either at query time or at update time.

❖ Evaluating Views (cont)

Example: Using the **InnerCityHotels** view.

```
CREATE VIEW InnerCityHotels AS
  SELECT name, license
  FROM Bars
  WHERE addr IN ('The Rocks', 'Sydney');
```

```
SELECT name
FROM InnerCityHotels
WHERE license = '123456';
```

--is **rewritten into** the following form before execution

```
SELECT name
FROM Bars
WHERE addr IN ('The Rocks', 'Sydney')
      AND license = '123456';
```

❖ Materialized Views

Materialized views are implemented as stored tables

On each update to base tables, need to also update the view table.

Clearly this costs space and makes updates more expensive.

However, in a situation where

- updates are infrequent compared to queries on the view
- the cost of "computing" the view is expensive

this approach provides substantial benefits.

Materialized views are used extensively in data warehouses.

Produced: 29 Sep 2020