CS 450

SQL - Views

Views

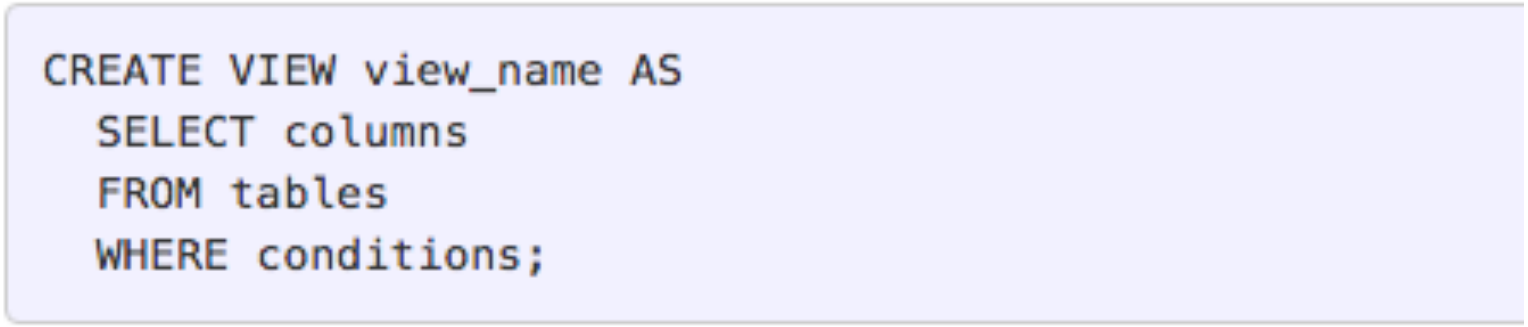
- In some cases, it is not desirable for all users to see the entire logical model (that is, all the actual relations stored in the database.)
- Consider a person who needs to know an instructors name and department, but not the salary. This person should see a relation described, in SQL, by

  select *ID*, *name*, *dept_name*
  from *instructor*

- A **view** provides a mechanism to hide certain data 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**.

View Definition

- A view is defined using the **create view** statement which has the form

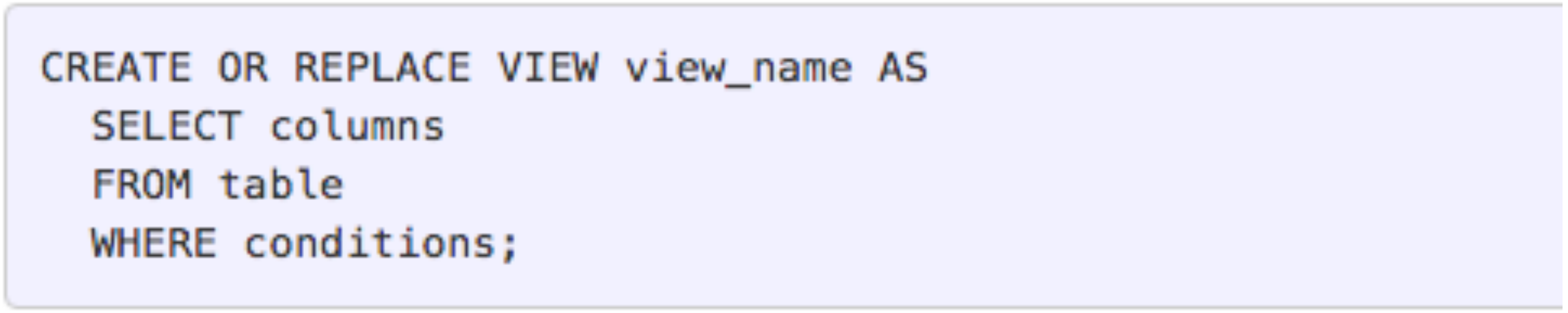


```
CREATE VIEW view_name AS
  SELECT columns
  FROM tables
  WHERE conditions;
```

- Once a view is defined, the view name can be used to refer to the virtual relation that the view generates.
- View definition is not the same as creating a new relation by evaluating the query expression
  - Rather, a view definition causes the saving of an expression; the expression is substituted into queries using the view.

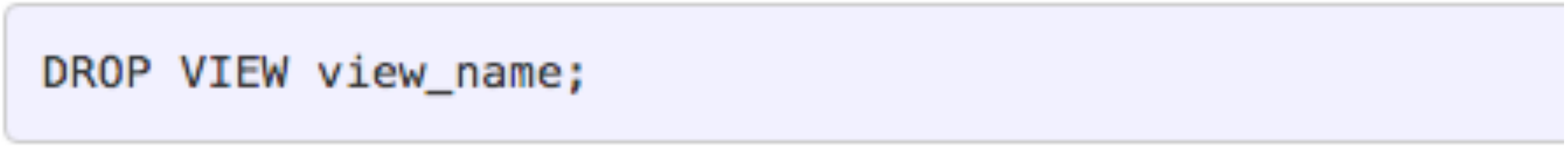
Update/Drop View Definition

- You can update view definition without dropping it first by using the CREATE OR REPLACE VIEW statement.



```
CREATE OR REPLACE VIEW view_name AS
  SELECT columns
  FROM table
  WHERE conditions;
```

- To drop a view



```
DROP VIEW view_name;
```

Example Views

- A view of instructors without their salary

  CREATE VIEW *faculty* **AS**
  SELECT *ID*, *name*, *dept_name*
  FROM *instructor*

- Find all instructors in the Biology department

  SELECT *name*
  FROM *faculty*
  WHERE *dept_name* = 'Biology'

- Create a view of department salary totals

  CREATE VIEW *departments_total_salary*(*dept_name*, *total_salary*) **AS**
  SELECT *dept_name*, **SUM** (*salary*)
  FROM *instructor*
  GROUP BY *dept_name*;

Views Defined Using Other Views

- **CREATE VIEW** *physics_fall_2009* **AS**
  SELECT *course.course_id*, *sec_id*, *building*, *room_number*
  FROM *course*, *section*
  WHERE *course.course_id* = *section.course_id*
  AND *course.dept_name* = ’Physics’
  AND *section.semester* = ’Fall’
  AND *section.year* = ’2009’;

- **CREATE VIEW** *physics_fall_2009_watson* **AS**
  SELECT *course_id*, *room_number*
  FROM *physics_fall_2009*
  WHERE *building*= ’Watson’;

View Expansion

- Expand use of a view in a query/another view

```
CREATE VIEW physics_fall_2009_watson AS
(SELECT course_id, room_number
FROM (SELECT course.course_id, building, room_number
          FROM course, section
          WHERE course.course_id = section.course_id
               AND course.dept_name = ’Physics’
               AND section.semester = ’Fall’
               AND section.year = ’2009’)
WHERE building= ’Watson’;
```

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.

View Expansion

- A way to define the meaning of views defined in terms of other views.
- Let view v_1 be defined by an expression e_1 that may itself contain uses of view relations.
- View expansion of an expression repeats the following replacement step:

```
repeat
   Find any view relation $v_i$ in $e_1$
   Replace the view relation $v_i$ by the expression defining $v_i$
until no more view relations are present in $e_1$
```

- As long as the view definitions are not recursive, this loop will terminate

Update Data in a View

- A view in Oracle is created by joining one or more tables. When you update record(s) in a view, it updates the records in the underlying tables that make up the view.
- e.g. Add a new tuple to *faculty* view which we defined earlier

  INSERT INTO *faculty* **VALUES** (' 30765' , ' Green' , ' Music');

  This insertion must be represented by the insertion of the tuple

  (' 30765' , ' Green' , ' Music' , null)

  into the *instructor* relation.

Some Updates cannot be Translated Uniquely

- **CREATE VIEW** *instructor_info* **AS**
  SELECT *ID*, *name*, *building*
  FROM *instructor*, *department*
  WHERE *instructor.dept_name*= *department.dept_name*;
- **INSERT INTO** *instructor_info* **VALUES** (' 69987', 'White', 'Taylor');
    - Which department, if multiple departments in Taylor?
    - What if no department is in Taylor?
- Most SQL implementations allow updates only on simple views
  – The **from** clause has only one database relation.
  – The **select** clause contains only attribute names of the relation, and does not have any expressions, aggregates, or **distinct** specification.
  – Any attribute not listed in the **select** clause can be set to null
  – The query does not have a **group** by or **having** clause.

And Some Not at All

- **CREATE VIEW** *history_instructors* **AS**
  SELECT *
  FROM *instructor*
  WHERE *dept_name*= ' History' ;
- What happens if we insert (' 25566' , ' Brown' , ' Biology' , 100000) into *history_instructors?*

What Happens If the Underlying Table(s) Are Dropped?

- The view continue to exist even after the table(s) that the view is based on are dropped.
- However, if we try to query the view after the table(s) have been dropped, we get a message saying that the view has errors.
- Once we recreate the table(s), the view is fine again.

Materialized Views

- **Materializing a view**: create a physical table containing all the tuples in the result of the query defining the view
- If relations used in the query are updated, the materialized view result becomes out of date
  - Need to **maintain** the view, by updating the view whenever the underlying relations are updated.

Transactions

- Unit of work
- Atomic transaction
  - either fully executed or rolled back as if it never occurred
- Isolation from concurrent transactions
- Transactions begin implicitly
  - Ended by **commit work** or **rollback work**
- But default on most databases: each SQL statement commits automatically
  - Can turn off auto commit for a session (e.g. using API)
  - In SQL:1999, can use: **begin atomic** …. **end**
    - Not supported on most databases