Views

What are views good for? (1)

- Simplifying complex queries: We saw one example. Here is another that allows the user to "pretend" that there is a single table in the database
  - CREATE VIEW SRB as

```
SELECT S.sid, sname, rating, age, R.bid, day,
          bname, color
FROM Sailors S, Boats B, Reserves R
WHERE S.sid = R.sid and R.bid = B.bid
```

What are views good for? (1)

- Now: Find snames of Sailors who reserved 103 using SRB

What are views good for? (2)

- Security issues – preventing unauthorized access. Example: hiding the rating value

```
CREATE VIEW SailorInfo
SELECT sname, sid, age
FROM Sailors

grant SELECT on SailorInfo to joe;
```

Changing Tables though a View

Changing a Table through a View

- If a view is based on a single table you can insert, update and delete rows from the table through the view, if you have the necessary permissions, under the following conditions:
  - You can't insert if the underlying table has non null columns not appearing in the view
  - You can't insert or update if any of the view columns referenced in the command contains functions or calculations
  - You can't insert, update or delete if the view contains group by or distinct.

What can be changed?

	Only Values of Rows seen through the View	Only Values of Columns seen through the View
Insert	No	Yes
Update	Yes	Yes
Delete	Yes	No

7

Inserting Allowed

```
CREATE VIEW OldSailors as
SELECT *
FROM Sailors
WHERE age > 50;
```

```
INSERT INTO OldSailors(sid,sname,age,rating)
VALUES(12, Joe ,51,10);
```

When we select from OldSailors next time, we will see Joe

8

Inserting Allowed

```
CREATE VIEW OldSailors as
SELECT *
FROM Sailors
WHERE age > 50;
```

```
INSERT INTO OldSailors(sid,sname,age,rating)
VALUES(12, Mary ,49,10);
```

When we select from OldSailors next time, we will **not** see Mary!

9

Preventing Insertions that are not seen through the View

```
CREATE VIEW OldSailors as
SELECT *
FROM Sailors
WHERE age > 50
WITH CHECK OPTION;
```

✔
```
INSERT INTO OldSailors(sid,sname,age,rating)
VALUES(12, Joe ,51,10);
```

✗
```
INSERT INTO OldSailors(sid,sname,age,rating)
VALUES(12, Mary ,49,10);
```

10

Preventing Insertions that are not seen through the View

- How can Oracle implement WITH CHECK OPTION?

11

Inserting Not Allowed

```
CREATE VIEW SailorsInfo as
SELECT sname, rating
FROM Sailors
WHERE age>50;
```

```
INSERT INTO SailorsInfo VALUES( Joe ,10);
```

⬆

Illegal!
Why?

12

Updating Allowed

```
CREATE VIEW SailorsInfo as
SELECT sname, rating, age
FROM Sailors
WHERE age>50;
```

```
UPDATE SailorsInfo
SET rating = 6
WHERE sname =  Joe ;
```

Oracle only changes the rating of Joes who are older than 50.

Implemented by adding WHERE condition of view to WHERE condition of Update

```
UPDATE Sailors
SET rating = 6
WHERE sname =  Joe  and age>50;
```

Updating Allowed

```
CREATE VIEW SailorsInfo2 as
SELECT sname, rating, age
FROM Sailors
WHERE age>50;
```

```
UPDATE SailorsInfo2
SET age = age   1;
```

How is it implemented?

Will cause tuples to "disappear from the view"

Can prevent this "WITH CHECK OPTION"

Updating Not Allowed

```
CREATE VIEW SailorsInfo3 as
SELECT sname, rating + age as ra
FROM Sailors
WHERE age>50;
```

```
UPDATE SailorsInfo3
SET ra = 7
WHERE sname =  Joe ;
```

Illegal!
Why?

Deleting Allowed

```
CREATE VIEW SailorsInfo3 as
SELECT sname, rating + age as ra
FROM Sailors
WHERE age>50;
```

```
DELETE FROM SailorsInfo3
WHERE sname =  Joe
        and ra = 56;
```

Oracle only deletes Joes visible through the view.
How do you think that this is implement by Oracle?

Some More Examples: What will these commands do?

```
CREATE VIEW OldSailors as
SELECT *
FROM Sailors
WHERE age > 50;
```

```
UPDATE OldSailors
SET rating = 10;
```

```
UPDATE OldSailors
SET age = age +1
WHERE age <= 50;
```

```
DELETE FROM OldSailors;
```

Materialized Views

What and Why?

- **What:** A materialized view is a view that actually exists as a table
- **Why:** This can be more efficient than re-computing the view's query each time it is accessed
- **Problem:** How is the materialized view kept up to date when the underlying tables are changed?
- **Note:** We will just see the ideas here and not the syntax

19

Simple vs. Complex

A Simple Materialized View:

1. Selects rows from only 1 table
2. Does not perform set operations, joins or group bys

Otherwise, it is a **Complex Materialized View**.

20

Options for Materialized Views

- Refresh mode, one of
  - Complete: Recompute the query
  - Fast (only possible for Simple Materialized Views): add minimal changes
- When is refresh performed?
  - On demand: When refresh is specifically asked for
  - On commit: When underlying table has changed

21

Fast Refresh

- Consider a materialized view defined by the query:

```
SELECT sname, age
FROM  Sailors
WHERE rating < 10 and age > 50;
```

22

Fast Refresh

- How would the materialized view be updated when the following commands are performed?

```
INSERT INTO Sailors(sid,sname,rating,age)
VALUES (12, Joe ,8,52);
```

```
DELETE FROM Sailors
WHERE age < 54;
```

```
UPDATE Sailors SET age = age  - 1
WHERE rating < 10;
```

23

Query Rewrite

- We can use a materialized view in a query, similarly to a table and to a regular view
- The Query Rewrite Option specifies that Oracle can automatically substitute a materialized view in a query, instead of a table mentioned in the query.
- Why is this useful?

24

Query Rewrite

- Suppose the materialized view from before was called InterestingSailors.
- Given the query:

```
SELECT age
FROM  Sailors
WHERE rating < 10 and age > 51
```

- Oracle could decide to evaluate instead

```
SELECT age
FROM  InterestingSailors
WHERE age > 51;
```

25

Null Values

26

Null Values in Expressions

- The result of an arithmetic expression, over something that is null -> is null (e.g., null*0, null = null)
- Nulls in logical expressions:
  - null AND true -> null
  - null AND false -> false
  - null OR true -> true
  - null OR false -> null
  - NOT (null) -> null
- Tuples only pass the WHERE condition if the WHERE evaluates to true (not to false or null)

27

Nulls in Aggregation Functions

- count(*): counts all rows (even rows that are all null values)
- count(A): counts non-null As. returns 0 if all As are null
- sum(A), avg(A), min(A), max(A)
  - ignore null values of A
  - if A only contains null value, the result is null

28

Distinct and Group By

- Rows are considered identical if they have all the same non-null values or both have null values in the same columns
- Distinct removes duplicates of such rows
- Such rows form a single group when using GROUP BY

29

Example

A

B	C
1	
2	
3	4
3	5

```
SELECT count(*), count(c), min(c), sum(c)
FROM  (SELECT c
          FROM A
        WHERE c IS NULL or c <> NULL
      GROUP BY c)
```

30