CSE – 302
Database Management System Sessional

VIEW

View

- In **database** theory, a **view** is the result set of a stored query on the data,

  which the **database** users can query just as they would in a persistent **database** collection object.

- This pre-established query command is kept in the **database** dictionary.

View

- Based on one or more table (base table) or another view
- Doesn’t contain data and is not allocated any storage space
- Result of a sub query that is given a name and can be referenced in a FROM Clause of a SELECT statement, like any table
- Display selectively data contained in other tables
- Can retrieve data from a view as you do with any table

Advantages of Using View

- Provide data independence: If extra columns are added to the original tables, the view does not need to be changed

- Views can be customized to meet user needs: Different users can see the data in different ways

- Views contain only those columns needed by the user

- A collection of tables used by the view will be viewed conceptually as a single table

Advantages of Using View

- Provide a measure of security
- Can be removed without affecting the underlying data
- Views can join and simplify multiple tables into a single virtual table.

Types of View

- Simple View
  - Derives data from only one table
  - No group function

- Complex View
  - Derives data from more than one table
  - Can include group function

View Creation

- **Syntax:**

  CREATE [OR REPLACE] [FORCE | NOFORCE] VIEW *viewname* (*columnname1, columnname2*) AS *subquery*

  [WITH CHECK OPTION [CONSTRAINT *constraintname*]]

  [WITH READ ONLY [CONSTRAINT *constraintname*]]

View Creation

CREATE [OR REPLACE] [FORCE | NOFORCE] VIEW *viewname* (*columnname1, columnname2* .. .)

- CREATES the view with the given name
- Name should not be another DB object in the current schema
- There is no way to modify an existing view
- Use OR REPLACE if view already exists

View Creation

CREATE [OR REPLACE] [FORCE | NOFORCE] VIEW *viewname* (*columnname*, .. .)

- (columnname, .. .) is used if you want to give different names (aliases) to the columns of the view, not those in the referenced table

- The number of names listed must match the number of columns returned by the subquery

- If column aliases are given in the subquery, the aliases will be used as the names of columns in the view

View Creation

CREATE [OR REPLACE] [FORCE | NOFORCE] VIEW *viewname* (*columnname*, ...)

- NOFORCE
  - Is the default
  - The view cannot be created if the referenced tables don't exist or currently unavailable

View Creation

CREATE [OR REPLACE] [FORCE | NOFORCE] VIEW *viewname* (*columnname*, .. .)

- FORCE
  - Creates the view even if the referenced table does not exist or the data is invalid.
  - Usually used when a new DB is being developed and the data have not been loaded into the DB

View Creation

- **Syntax:**

  CREATE [OR REPLACE] [FORCE | NOFORCE] VIEW viewname (columnname..) AS Subquery

  [WITH CHECK OPTION [CONSTRAINT constraintname]]

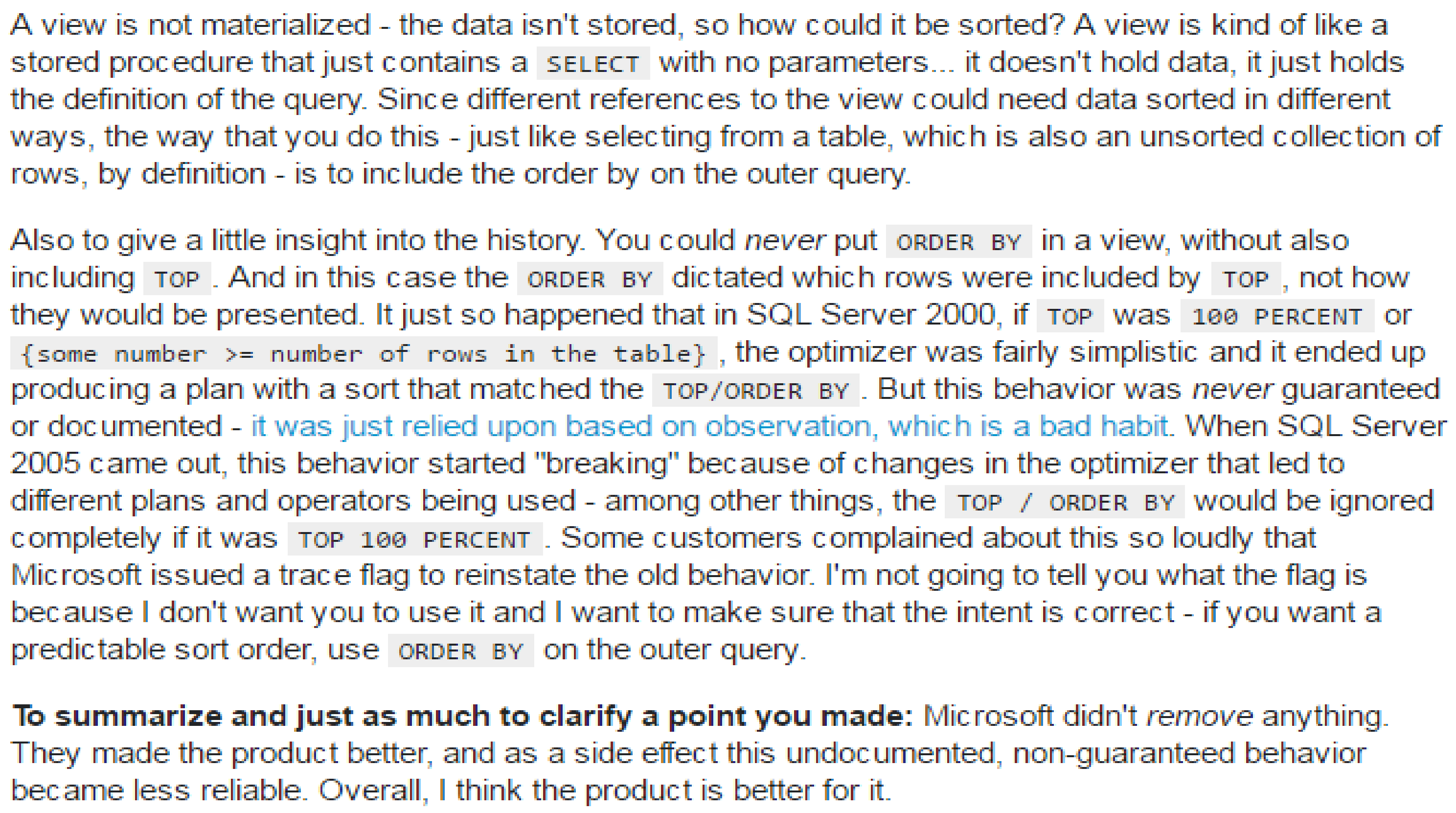
  [WITH READ ONLY [CONSTRAINT constraintname]]

View Creation

AS clause:

- The subquery must be a complete subquery
- It can reference more than one table, can include single-row & multiple-row functions, WHERE, GROUP BY, and nested subqueries
- The subquery cannot include ORDER BY clause

Why does ORDER BY not belong in a View?



A view is not materialized - the data isn't stored, so how could it be sorted? A view is kind of like a stored procedure that just contains a `SELECT` with no parameters... it doesn't hold data, it just holds the definition of the query. Since different references to the view could need data sorted in different ways, the way that you do this - just like selecting from a table, which is also an unsorted collection of rows, by definition - is to include the order by on the outer query.

Also to give a little insight into the history. You could *never* put `ORDER BY` in a view, without also including `TOP`. And in this case the `ORDER BY` dictated which rows were included by `TOP`, not how they would be presented. It just so happened that in SQL Server 2000, if `TOP` was `100 PERCENT` or `{some number >= number of rows in the table}`, the optimizer was fairly simplistic and it ended up producing a plan with a sort that matched the `TOP/ORDER BY`. But this behavior was *never* guaranteed or documented - it was just relied upon based on observation, which is a bad habit. When SQL Server 2005 came out, this behavior started "breaking" because of changes in the optimizer that led to different plans and operators being used - among other things, the `TOP / ORDER BY` would be ignored completely if it was `TOP 100 PERCENT`. Some customers complained about this so loudly that Microsoft issued a trace flag to reinstate the old behavior. I'm not going to tell you what the flag is because I don't want you to use it and I want to make sure that the intent is correct - if you want a predictable sort order, use `ORDER BY` on the outer query.

To summarize and just as much to clarify a point you made: Microsoft didn't *remove* anything. They made the product better, and as a side effect this undocumented, non-guaranteed behavior became less reliable. Overall, I think the product is better for it.

Ref: http://dba.stackexchange.com/questions/21434/why-does-order-by-not-belong-in-a-view

WITH CHECK OPTION Constraint

- Limited usage
  - Check referential integrity through the view
  - Enforce constraints at the DB level: INSERTS & UPDATES performed through the view can’t create rows which the view can't select

WITH CHECK OPTION Constraint

- If used, prevents data changes that will make the data subsequently inaccessible to the view

- Ensures that any valid DML is allowed, even if the result is that the row(s) being changed would no longer be included in the view

- CUSTOMER (cust_id, cust_name, cust_dob, cust_city, cust_street)
- ACCOUNT (account_id, balance, type)
- LOAN (loan_id, amount)
- BRANCH (branch_name, branch_city, assets)
- BORROWER (cust_id, loan_id)
- DEPOSITOR (cust_id, account_id)

Example1: Simple View Creation

```
CREATE VIEW CUSTOMER_VIEW
("Customer ID", "Name", "Date of Birth",
"City") AS
SELECT Cust_id, Cust_name, Cust_dob,
Cust_city FROM CUSTOMER
```

Example2: Modify Existing View

```
CREATE OR REPLACE VIEW
CUSTOMER_VIEW ("Customer ID", "Name",
"Date of Birth", "City") AS
SELECT Cust_id, Cust_name, Cust_dob,
Cust_city FROM CUSTOMER
```

Example3: Use Contraints

```
CREATE  OR REPLACE VIEW
CUSTOMER_VIEW ("Customer ID", "Name",
"Date of Birth", "City") AS
SELECT Cust_id, Cust_name, Cust_dob,
Cust_city FROM CUSTOMER WHERE
CUST_DOB> to_date('01-JAN-1975', 'DD-
MON-YYYY')
WITH CHECK OPTION CONSTRAINT
Birthday_Check;
```

Example3: Use Contraints (cont..)

INSERT INTO *CUSTOMER_VIEW* VALUES ('C000000110', 'Sadia', to_date('12-04-1990','DD-MM-YYYY'),'Dhaka');

1 ROW INSERTED.

Example3: Use Contraints (cont..)

INSERT INTO *CUSTOMER_VIEW* VALUES ('C000000110', 'Sadia', to_date('12-04-1890','DD-MM-YYYY'),'Dhaka');

Error report -

SQL Error: ORA-01402: view **WITH CHECK OPTION** where-clause violation

Example4:

```
CREATE  OR REPLACE VIEW
CUSTOMER_VIEW ("Customer ID", "Name",
"Date of Birth", "City") AS
SELECT Cust_id, Cust_name, Cust_dob,
Cust_city FROM CUSTOMER WHERE
CUST_DOB> to_date('01-JAN-1975', 'DD-
MON-YYYY')
WITH READ ONLY CONSTRAINT
View_Read_Only;
```

Example4:

```
INSERT INTO CUSTOMER_VIEW VALUES
    ('C000000110', 'Sadia', to_date ('12-04-1890',
    'DD-MM-YYYY'), 'Dhaka');
```

```
SQL Error: ORA-42399: cannot perform a DML
    operation on a read-only view
```

- CUSTOMER (cust_id, cust_name, cust_dob, cust_city, cust_street)
- ACCOUNT (account_id, balance, type)
- LOAN (loan_id, amount)
- BRANCH (branch_name, branch_city, assets)
- BORROWER (cust_id, loan_id)
- DEPOSITOR (cust_id, account_id)

Example5: Complex View Creation

```
CREATE OR REPLACE VIEW
CUSTOMER_VIEW ("Customer ID", "Name",
"Date of Birth", "City", "Account ID",
"Balance") AS
SELECT Cust_id, Cust_name, Cust_dob,
Cust_city, Account_id, Balance
FROM  CUSTOMER JOIN DEPOSITOR
USING(CUST_ID) JOIN ACCOUNT USING
(ACCOUNT_ID)
WHERE balance> 500;
```

INSERT Query on Simple View

INSERT INTO *CUSTOMER_VIEW* VALUES ('C000000110', 'Sadia', to_date ('12-04-1890', 'DD-MM-YYYY'), 'Dhaka');

- Column name SHOULD BE the View's column name, NOT the actual table's column name

SELECT Query on View

```
SELECT "Name", "Balance" FROM
CUSTOMER_VIEW WHERE "Date of Birth"
> to_date('01-01-1980','DD-MM-YYYY');
```

- Column name SHOULD BE the View's column name, NOT the actual table's column name

Dropping a View

DROP VIEW *view_name*

References

1. Oracle_Database_11g_The_Complete Reference
2. https://en.wikipedia.org/wiki/View_(SQL)
3. http://dba.stackexchange.com/
4. http://www.w3schools.com/sql
5. Book: Database System Concepts written by Avi Silberschatz, Henry F. Korth, S. Sudarshan

Thank You