Views in SQL

Material covered from Chapter 8, A First Course in Database Systems by Jenifer and Garcia &

Stanford course of Database Management and Data Systems, Spring 2023 – CS145

Definition and Key Points

- A view in SQL terminology is a single table that is derived from other **tables**.
- Tables can be **base tables** (Stored relations) or previously defined **views.**
- A view does not necessarily exists in physical form. It is considered to be a **virtual table.**
- **Virtual table/View's definition** is stored in database. It's tuples are not.

Definition taken from the book Fundamentals of Database System by Elmasri

Syntax (ADD/DELETE)

CREATE VIEW <view-name> AS (<view-definition>)

Title	Year	Genre	studioName	producer #
X	1992	Comedy	Paramount	123
Y	2002	Action	B	456
Z	2019	Horror	Paramount	789

```
Movies(title, year, length, genre, studioName, producerC#)

1)  CREATE VIEW ParamountMovies AS
2)      SELECT title, year
3)      FROM Movies
4)      WHERE studioName = 'Paramount';
```

Title	Year
X	1992
Z	2019

DROP VIEW <view-name>

Dropping view will not delete tuples of base tables

Example taken from the book, A first course in database systems by Ullman

Example 1

- We will frequently refer to COMPANY database for issuing queries that retrieve employ name and the project names that the employee works on.

Schema:

EMPLOYEE(fname, lname, ssn, bdate, address, gender, salary, super_ssn,dno)

DEPARTMENT(dname, dnumber, mgr_ssn, mgr_start_date)

DEPTLOCATIONS(dnumber, dlocation)

PROJECT(name, pnumber, plocation, dnum)

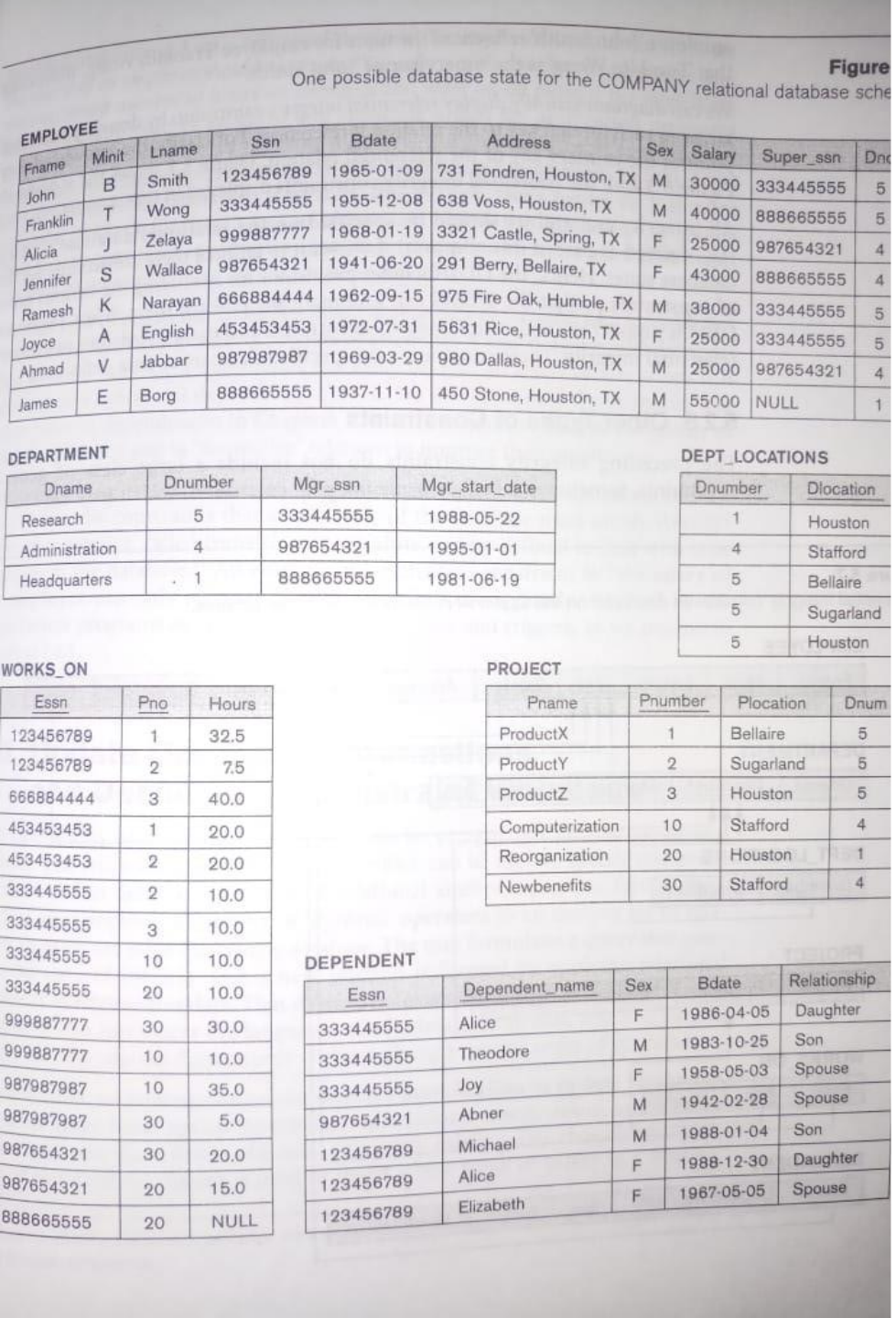
WORKS_ON(essn, pno, hours)

DEPENDENT(essn, dependent_name, gender, bdate, relationship)

```
CREATE VIEW WORKS_ON1 AS
    SELECT fname, lname, pname, hours
    FROM EMPLOYEE, PROJECT, WORKS_ON
`   WHERE ssn=essn AND pno=pnumber
```

```
CREATE VIEW DEPT_INFO(dept_name,
no_of_emps, total_sal) AS
    SELECT dname, COUNT(*), SUM(salary)
    FROM DEPARTMENT, EMPLOYEE
`   WHERE Dnumber=Dno
    GROUP BY Dname
```

Definition taken from the book Fundamentals of Database System by Elmasri



Figure

One possible database state for the COMPANY relational database sche

EMPLOYEE

Fname	Minit	Lname	Ssn	Bdate	Address	Sex	Salary	Super_ssn	Dno
John	B	Smith	123456789	1965-01-09	731 Fondren, Houston, TX	M	30000	333445555	5
Franklin	T	Wong	333445555	1955-12-08	638 Voss, Houston, TX	M	40000	888665555	5
Alicia	J	Zelaya	999887777	1968-01-19	3321 Castle, Spring, TX	F	25000	987654321	4
Jennifer	S	Wallace	987654321	1941-06-20	291 Berry, Bellaire, TX	F	43000	888665555	4
Ramesh	K	Narayan	666884444	1962-09-15	975 Fire Oak, Humble, TX	M	38000	333445555	5
Joyce	A	English	453453453	1972-07-31	5631 Rice, Houston, TX	F	25000	333445555	5
Ahmad	V	Jabbar	987987987	1969-03-29	980 Dallas, Houston, TX	M	25000	987654321	4
James	E	Borg	888665555	1937-11-10	450 Stone, Houston, TX	M	55000	NULL	1

DEPARTMENT

Dname	Dnumber	Mgr_ssn	Mgr_start_date
Research	5	333445555	1988-05-22
Administration	4	987654321	1995-01-01
Headquarters	1	888665555	1981-06-19

DEPT_LOCATIONS

Dnumber	Dlocation
1	Houston
4	Stafford
5	Bellaire
5	Sugarland
5	Houston

WORKS_ON

Essn	Pno	Hours
123456789	1	32.5
123456789	2	7.5
666884444	3	40.0
453453453	1	20.0
453453453	2	20.0
333445555	2	10.0
333445555	3	10.0
333445555	10	10.0
333445555	20	10.0
999887777	30	30.0
999887777	10	10.0
987987987	10	35.0
987987987	30	5.0
987654321	30	20.0
987654321	20	15.0
888665555	20	NULL

PROJECT

Pname	Pnumber	Plocation	Dnum
ProductX	1	Bellaire	5
ProductY	2	Sugarland	5
ProductZ	3	Houston	5
Computerization	10	Stafford	4
Reorganization	20	Houston	1
Newbenefits	30	Stafford	4

DEPENDENT

Essn	Dependent_name	Sex	Bdate	Relationship
333445555	Alice	F	1986-04-05	Daughter
333445555	Theodore	M	1983-10-25	Son
333445555	Joy	F	1958-05-03	Spouse
987654321	Abner	M	1942-02-28	Spouse
123456789	Michael	M	1988-01-04	Son
123456789	Alice	F	1988-12-30	Daughter
123456789	Elizabeth	F	1967-05-05	Spouse

VIEW – WORKS_ON1

WORKS_ON1:

Fname	Lname	Pname	Hours
John	Smith	ProductX	32.5
John	Smith	ProductY	7.5
Franklin	Wong	ProductY	10.0
Franklin	Wong	ProductZ	10.0
..	..	..	..

Querying Views

- We want to know names of all employees who work for project "ProductX"

```
SELECT Fname, Lname
FROM WORKS_ON1
WHERE Pname = 'ProductX'
```

Replace each view by a subquery that is identical to the view definition

```
SELECT Fname, Lname
FROM (SELECT fname, lname, pname, hours
     FROM EMPLOYEE, PROJECT, WORKS_ON
`    WHERE ssn=essn AND pno=pnumber)
WHERE Pname = 'ProductX
```

Are views updatable?

Issue: How will we translate the insert/update/delete query on VIEW to the actual base tables?

Updatable views: Queries are translated into base tables. INSTEAD OF triggers can also be used. (**Limited circumstances**)

SQL rules for updating views

- USE **SELECT** not **SELECT DISTINCT** for defining views.
- The **WHERE** clause must not involve **R** in a subquery.
- The **FROM** clause can only consist of one occurrence of **R** and no other relation
- The list in the **SELECT** clause must include enough attributes that for every tuple inserted into the view, we can fill the other attributes out with **NULL** values or the proper default. It is not permitted to project out an attribute that is declared **NOT NULL** and has no default.

SQL rules for updating views (same)

- A view with a single defining table is updatable if the view attributes contain the primary key of the base relation, as well as all attributes with the NOT NULL constraint that do not have default values specified.
- Views defined on multiple tables using joins are generally not updatable.
- Views defined using grouping and aggregate functions are not updatable.

Inserting in updatable view

```
Movies(title, year, length, genre, studioName, producerC#)

 1)  CREATE VIEW ParamountMovies AS
 2)      SELECT title, year
 3)      FROM Movies
 4)      WHERE studioName = 'Paramount';
```

```
INSERT INTO ParamountMovies
VALUES('Star Trek', 1979)
```

Will tuple be inserted?

Issue?

Example taken from the book, A first course in database systems by Ullman

Inserting in updatable view

- Insert into WORKS_ON1 view.

Schema:

EMPLOYEE(fname, lname, ssn, bdate, address, gender, salary, super_ssn,dno)

DEPARTMENT(dname, dnumber, mgr_ssn, mgr_start_date)

DEPTLOCATIONS(dnumber, dlocation)

PROJECT(name, pnumber, plocation, dnum)

WORKS_ON(essn, pno, hours)

DEPENDENT(essn, dependent_name, gender, bdate, relationship)

```
CREATE VIEW WORKS_ON1
AS SELECT Fname, Lname, Pname, Hours
     FROM EMPLOYEE, PROJECT, WORKS_ON
`    WHERE ssn=essn AND pno=pnumber
```

```
INSERT INTO WORKS_ON1
VALUES ('Kinza', 'Fatima','Zong', 20)
```

Is it valid?

Definition taken from the book Fundamentals of Database System by Elmasri

Deleting from a updatable view

```
DELETE FROM ParamountMovies
WHERE title LIKE '%Trek%'
```

Actual translation:

```
DELETE FROM Movies
WHERE title LIKE '%Trek%' AND studioName = 'Paramount'
```

Updating an updatable view

```
UPDATE ParamountMovies
SET YEAR = 1979
WHERE title = 'Star Trek the Movie'
```

Actual translation:

```
UPDATE Movies
SET year = 1979
WHERE title = 'Star Trek the Movie' AND studioName = 'Paramount'
```

View update issues

Schema:

EMPLOYEE(fname, lname, ssn, bdate, address, gender, salary, super_ssn,dno)

DEPARTMENT(dname, dnumber, mgr_ssn, mgr_start_date)

DEPTLOCATIONS(dnumber, dlocation)

PROJECT(name, pnumber, plocation, dnum)

WORKS_ON(essn, pno, hours)

DEPENDENT(essn, dependent_name, gender, bdate, relationship)

```
CREATE VIEW WORKS_ON1
AS SELECT Fname, Lname, Pname, Hours
     FROM EMPLOYEE, PROJECT, WORKS_ON
`    WHERE ssn=essn AND pno=pnumber
```

```
UPDATE WORKS_ON1
SET Pname = 'ProductY'
WHERE Lname = 'Smith' AND Fname = 'John'
AND Pname = 'ProductX'
```

Definition taken from the book Fundamentals of Database System by Elmasri

View update issues

```
UPDATE WORKS_ON1
SET Pname = 'ProductY'
WHERE Lname = 'Smith' AND Fname = 'John' AND Pname = 'ProductX'
```

Translation Possibility # 1:

```
UPDATE WORKS_ON
SET pno = (SELECT pnumber FROM PROJECT WHERE pname = ''ProductY')
WHERE essn IN (SELECT ssn FROM EMPLOYEE
                    WHERE Lname = 'Smith' AND Fname = 'John')
AND pno = (SELECT pnumber FROM PROJECT WHERE pname = 'ProductX'
```

Translation Possibility # 2:

```
UPDATE PROJECT SET pname = 'ProductY  WHERE pname = 'ProductX'
```

Definition taken from the book Fundamentals of Database System by Elmasri

View update issues

Schema:

EMPLOYEE(fname, lname, ssn, bdate, address, gender, salary, super_ssn,dno)

DEPARTMENT(dname, dnumber, mgr_ssn, mgr_start_date)

DEPTLOCATIONS(dnumber, dlocation)

PROJECT(name, pnumber, plocation, dnum)

WORKS_ON(essn, pno, hours)

DEPENDENT(essn, dependent_name, gender, bdate, relationship)

```
CREATE VIEW DEPT_INFO(dept_name,
no_of_emps, total_sal) AS
    SELECT dname, COUNT(*), SUM(salary)
    FROM DEPARTMENT, EMPLOYEE
`   WHERE Dnumber=Dno
    GROUP BY Dname
```

Definition taken from the book Fundamentals of Database System by Elmasri

```
UPDATE DEPT_INFO
SET total_sal = 10000
WHERE dname = 'Research'
```

It does not make sense.

Key Points

- View is not realized at the time of view definition but rather at the time we specify a query on the view.
- Views can be queried. Are they updatable? (in some cases)
- Views can be materialized, constructed periodically from database and stored there (**materialized views)**

Materialized views

Advantages of view

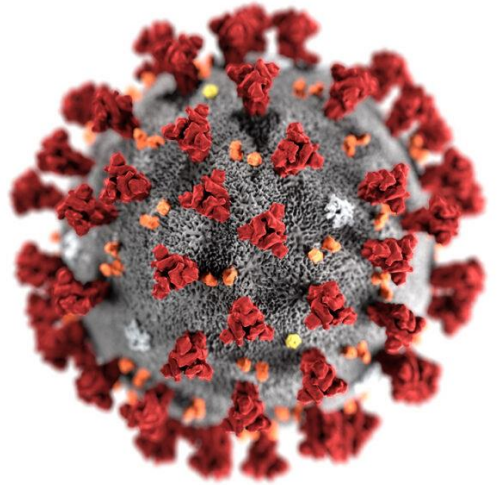
- Abstract view
- Hides complexity
- Limits exposure to outer world
- Simplify queries
- Security

View to materialized view

Need for materialized view:

- Frequent complex queries
- Batch processes
- Analysis on data

Definition

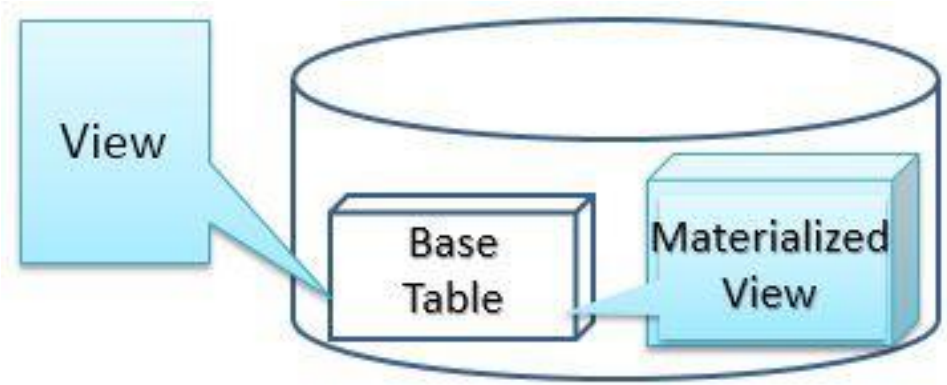


- Views are logical description of base tables
- If they are frequently used, we can *materialize* it.
- Cost is associated with maintaining materialized views.
- If underlying base tables changes, we must recompute parts of the *materialized views.*
- For simple views, it is possible to limit the number of times we need to consider changing the materialized view.

Definition taken from the book A First Course in Database Systems
Image: https://www.statnews.com/2020/02/11/disease-caused-by-the-novel-coronavirus-has-name-covid-19/

Definition

- Change in attributes of underlying tables, that are not defined in materialized view, will not effect the materialized view
- Change in other relations, that are not used to define materialized view, will not effect the materialized view



Definition taken from the book A First Course in Database Systems
Image: https://techdifferences.com/difference-between-view-and-materialized-view.html

Syntax (ADD/DELETE)

```
CREATE MATERIALIZED VIEW <view-name> AS (<view-definition>)
```

```
DROP VIEW <view-name>
```

Schema:

EMPLOYEE(nid, empid, name, gender, bdate, married)

JOB(jid, title, description, nhours)

EMPJOB(empid, jid, starttime, endtime, managerid)

```
CREATE MATERIALIZED VIEW EmployeeJob AS
        SELECT e.name, j.title, ej.starttime, ej.endtime
        FROM EMPLOYEE e, JOB j, EMPJOB ej
        WHERE e.empid = ej.empid AND j.jobid = ej.jid
```

Handling modifications in base tables (Insertion)

Whenever underlying base table changes, should materialized view be re-constructed? Or is there any other option?

```
CREATE MATERIALIZED VIEW EmployeeJob AS
        SELECT e.name, j.title, ej.starttime, ej.endtime
        FROM EMPLOYEE e, JOB j, EMPJOB ej
        WHERE e.empid = ej.empid AND j.jobid = ej.jid
```

Insertion in underlying base tables:

INSERT INTO EMPLOYEE

VALUES('3543212345678', 'XX23', 'Ali', 'M', '23-12-1990', 'Y')

INSERT INTO EMPJOB('XX23', 'JJ48', '04-04-2020', '')

DBMS will automatically do the following:

SELECT title FROM JOB WHERE jid = 'JJ48'

INSERT INTO EmployeeJob

VALUES('Ali', 'Data analysist', '04-04-2020', '')

Schema:

EMPLOYEE(cnic, empid, name, gender, bdate, married)

JOB(jid, title, description, nhours)

EMPJOB(empid, jid, starttime, endtime, managerid)

Handling modifications in base tables (Deletion)

Whenever underlying base table changes, should materialized view be re-constructed? Or is there any other option?

```
CREATE MATERIALIZED VIEW EmployeeJob AS
        SELECT e.name, j.title, ej.starttime, ej.endtime
        FROM EMPLOYEE e, JOB j, EMPJOB ej
        WHERE e.empid = ej.empid AND j.jobid = ej.jid
```

Deletion from underlying base tables:

DELETE FROM EMPJOB WHERE EMPID = 'XX23'

DBMS will automatically do the following:

DELETE FROM EmployeeJob

WHERE empid = 'XX23'

Schema:

EMPLOYEE(cnic, empid, name, gender, bdate, married)

JOB(jid, title, description, nhours)

EMPJOB(empid, jid, starttime, endtime, managerid)

Key points

- Changes to materialized views are incremental
- Insertion, deletions and updates to a base table can be implemented by a small number of queries to the base tables followed by modifications statements on the materialized views.

Periodic Maintenance of Materialized Views

- Example: Departmental store maintains database for inventory. It changes with every sale.
- Analysis and pattern study on data
- Analysis works good on aggregate data
- When modifications dominate, it is costly to have materialized views or even indexes, on the data
- **Solution:**
- Do not keep them up-to-date as the base table change
- Update each night/any other period of time given application

Materialized Views in Queries

- Can be referred in the FROM clause

```
SELECT * FROM materialized_myview
```

- Effect of replacing base tables in queries by materialized view
  - Makes query faster

Can we replace base tables in queries by materialized view?

YES, but with certain rules.

Replacing Base Tables with Materialized Views in Queries

M. view V:

SELECT L_v

FROM R_v

WHERE C_v

L_v is the list of attributes

R_v is the list of relations

C_v is a condition

Query Q:

SELECT L_Q

FROM R_Q

WHERE C_Q

L_Q is the list of attributes

R_Q is the list of relations

C_Q is a condition

Conditions:

- The relations in list R_v all appear in the list R_Q
- The condition C_Q is equivalent to C_v AND C for some condition C.
- If C is needed, then the attributes of relations on list R_v that C mentions are the attributes on the list L_v
- Attributes on the list L_Q that come from relations on the list R_v are also on the list L_v

Rewriting query (if condition meets):

a) Replace the list R_Q by V and the relations that are on list R_Q but not on R_v

b) Replace C_Q by C. If C is not needed (i.e., $C_v = C_Q$), then there is no WHERE clause.

Example

Movies(title, year, genre,studioName, producer#)

StarsIn(movieTitle, movieYear, starName)

MovieExec(name, address, cert#, netWorth)

Materialized view V:

```
CREATE MATERIALIZED VIEW MovieProd AS
SELECT title, year, name
FROM Movies, MoviesExec
WHERE producer# = cert#
```

Query Q:

```
SELECT starName
FROM StarsIn, Movies, MoviesExec
WHERE movieTitle = title AND movieYear =
year AND producer# = cert# AND name = 'Max
Bialystock'
```

1. The relation in the FROM clause of V are all in the FROM clause of Q.
2. The condition from Q can be written as the condition from V AND C, where C =

movieTitle = title AND movieYear = year AND name = 'Max Bialystock'

3. The attribute of C that come from relations of V (Movies and MovieExec) are title, year and name. These attributes all appear in the SELECT clause of V
4. No attributes from the SELECT list of Q is from a relation that appears in the FROM list of V.

Rewritten Query Q:

```
SELECT starName
FROM StarsIn, MovieProd
WHERE movieTitle = title AND movieYear = year
AND name = 'Max Bialystock
```

Automatic Creation of Materialized Views

- Establish query workload/observe DBMS query logs/Application programs that use database
- Automated materialized view selection advisor needs to generate candidate views (**difficult job – Why?**)

Candidate view:

1. Have a list of relations in the FROM clause that is a subset of those in the FROM clause of at least one query of the workload.
2. Have a WHERE clause that is the AND of conditions that each appear in at least one query.
3. Have a list of attributes in the SELECT clause that is sufficient to be used in at least one query.

- Query optimizer: Evaluate running time both with and without materialized views
- **Issue:** Index: Smaller – Less Space, Materialized views: Vary in size (multiple joins may be involved)

Evaluation:

$$\text{Benefit} = \frac{\text{Average running time of the query workload}}{\text{amount of space the view occupies}}$$

Triggers on Views

Triggers in View

- **Syntax:**

INSTEAD OF is used in place of **BEFORE** or **AFTER**

Action of trigger is performed **instead of** event itself.

Definition taken from the book Fundamentals of Database System by Elmasri

Example – Insert on view

Books(doi, title, year, pages, category, author, edition)

```
CREATE VIEW FictionBooks AS
        SELECT title, year
        FROM Books
        WHERE category = 'Fiction'
```

How to insert on View?

```
INSERT INTO FictionBooks
        VALUES('The Laughing Monsters, Denis Johnson', 2014)
```

What is the value of category?

Example – Insert on view with Trigger

Books(doi, title, year, pages, category, author, edition)

```
CREATE TRIGGER FictionBookInsert AS
INSTEAD OF INSERT ON FictionBooks
REFERENCING NEW ROW AS NewRow
FOR EACH ROW
INSERT INTO Books(title, year, category)
VALUES(NewRow.title, NewRow.year, 'Fiction')
```