CSE 301 – DATABASE

Lecture 9

Chapter 3: Relational Model

Modification of Database

- The content of the database may be modified using the following operations:
  - ✓ Insertion
  - ✓ Deletion
  - ✓ Updating
- All the operations are expressed using the assignment operator ($\leftarrow$).
- E.g., $R_{new} \leftarrow$ **operations on** (R_{old})

Deletion

- A delete request is expressed similarly to a query, expect instead of displaying tuples to the user, the selected tuples are removed from the database.
- In deletion, tuples are deleted from the relation
- Can delete only whole tuples; cannot delete values only particular attributes
- A deletion is expressed in relational algebra by: $R \leftarrow R - E$
  - ✓ Where R is a relation and E is a relational algebra expression.

Examples of Deletion

A	B	C
1	1	10
1	2	10
2	3	10
2	4	20

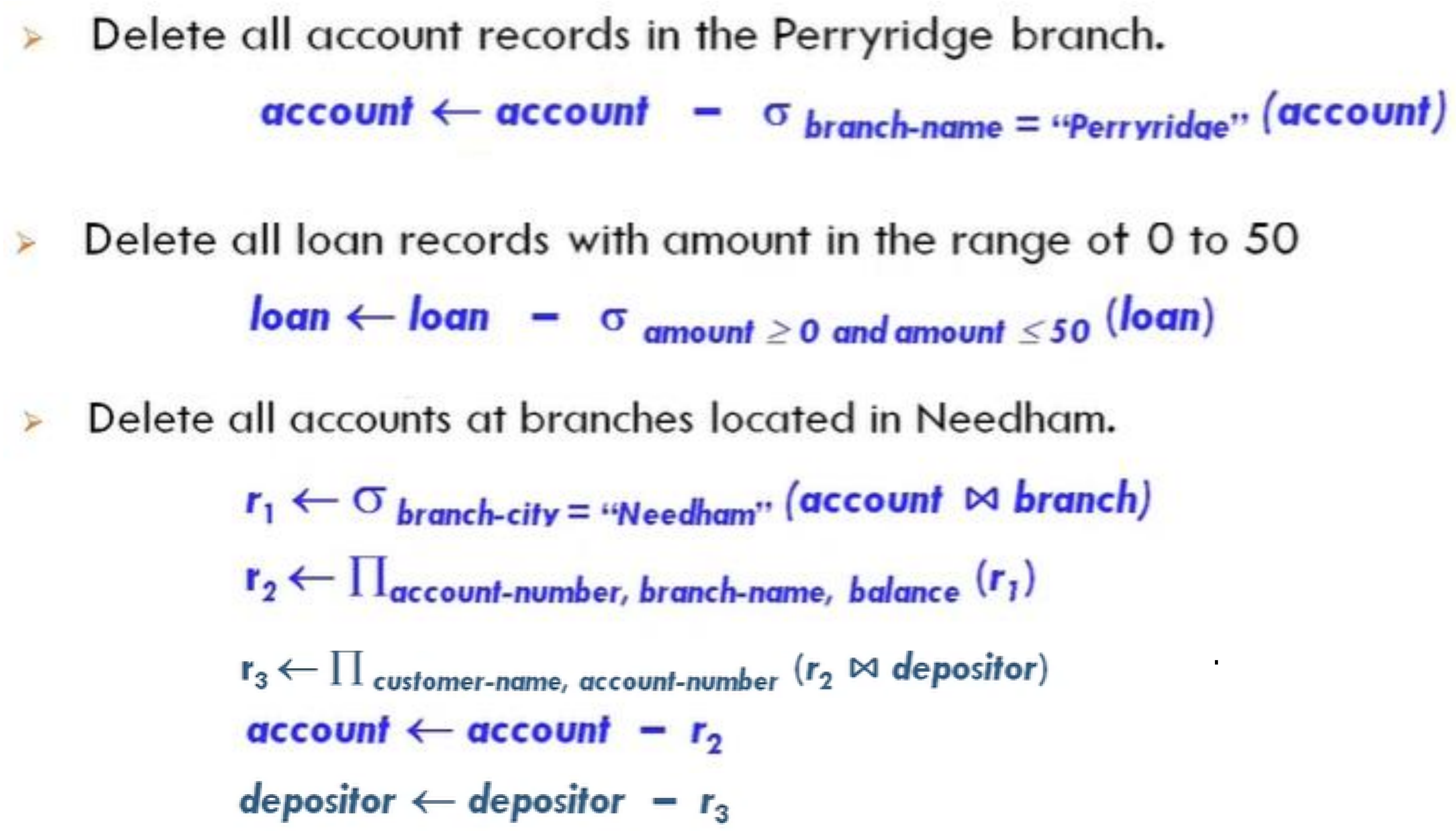
$r \leftarrow r - \sigma_{A=2}(r)$

A	B	C
1	1	10
1	2	10

$r \leftarrow r - \{(1, 2, 10)\}$

A	B	C
1	1	10
2	3	10
2	4	20

Examples of Deletion



- Delete all account records in the Perryridge branch.

$$account \leftarrow account - \sigma_{branch\text{-}name = \text{"Perryridge"}}(account)$$

- Delete all loan records with amount in the range of 0 to 50

$$loan \leftarrow loan - \sigma_{amount \geq 0 \text{ and } amount \leq 50}(loan)$$

- Delete all accounts at branches located in Needham.

$$r_1 \leftarrow \sigma_{branch\text{-}city = \text{"Needham"}}(account \bowtie branch)$$

$$r_2 \leftarrow \Pi_{account\text{-}number,\ branch\text{-}name,\ balance}(r_1)$$

$$r_3 \leftarrow \Pi_{customer\text{-}name,\ account\text{-}number}(r_2 \bowtie depositor)$$

$$account \leftarrow account - r_2$$

$$depositor \leftarrow depositor - r_3$$

Insertion

- ***Similar to deletion***, but use union operator (U) instead of minus (-) operator.
- In insertion, tuples are added to the relation
- To insert data into a relation, we either:
  - ✓ Specify a tuple to be inserted, or
  - ✓ Write a query whose result is asset of tuples to be inserted
- An insertion is expressed in relational algebra by: $R \leftarrow R \cup E$, where R is a relation and E is a relational algebra expression.
- The ***insertion of a single tuple*** is expressed by letting E be a constant relation containing one tuple.

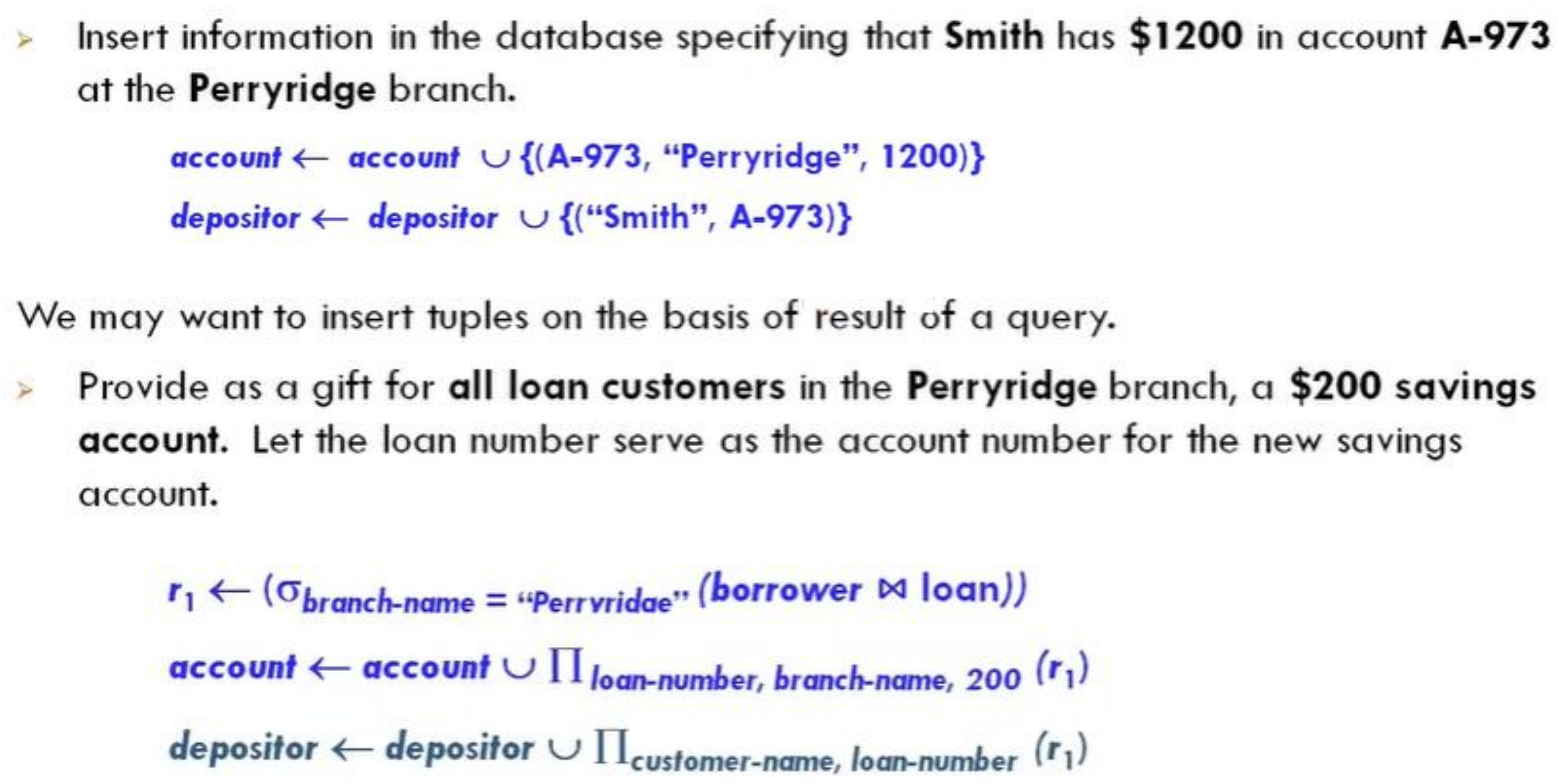
Examples of Insertion

A	B	C
1	1	10
1	2	10
2	3	10

$R \leftarrow R \cup \{(2, 4, 20)\}$

A	B	C
1	1	10
1	2	10
2	3	10
2	4	20

Examples of Insertion



- Insert information in the database specifying that **Smith** has **$1200** in account **A-973** at the **Perryridge** branch.

  $\textit{account} \leftarrow \textit{account} \cup \{(\text{A-973}, \text{“Perryridge”}, 1200)\}$

  $\textit{depositor} \leftarrow \textit{depositor} \cup \{(\text{“Smith”}, \text{A-973})\}$

We may want to insert tuples on the basis of result of a query.

- Provide as a gift for **all loan customers** in the **Perryridge** branch, a **$200 savings account.** Let the loan number serve as the account number for the new savings account.

  $r_1 \leftarrow (\sigma_{\textit{branch-name} = \text{“Perryridge”}}\,(\textit{borrower} \bowtie \textit{loan}))$

  $\textit{account} \leftarrow \textit{account} \cup \Pi_{\textit{loan-number, branch-name}, 200}\,(r_1)$

  $\textit{depositor} \leftarrow \textit{depositor} \cup \Pi_{\textit{customer-name, loan-number}}\,(r_1)$

Updating

- ***Updating*** is a mechanism to change a value in a tuple without changing all values in the tuple.
- Use the generalized projection operator to do this task.

$$R \leftarrow \Pi_{F1,\ F2,\ \ldots,\ Fn}(R)$$

- Each Fi can be either :
  - The attribute of R, or
  - An expression, involving only ***constants and the attributes*** of R, which gives the new value for the attribute.

Note: The schema of expression resulting from the generalized projection expression must match the original schema of R.

Examples of Updating

A	B	C
1	1	10
1	2	10
2	4	20

$\Rightarrow$

$R \leftarrow \Pi_{A,\ 2*B,\ C}(R)$

A	B	C
1	2	10
1	4	10
2	8	20

$R \leftarrow \Pi_{A,\ 2*B,\ C}(\sigma_{A=1}(R))$

A	B	C
1	2	10
1	4	10

Examples of Updating

- Make interest payments by increasing all balances by 5 percent.

$$account \leftarrow \Pi_{account\text{-}number,\ branch\text{-}name,\ balance * 1.05}(account)$$

- Pay all accounts with balances over \$10,000 a 6 percent interest and pay all others 5 percent

$$\begin{aligned} account \leftarrow\ & \Pi_{account\text{-}number,\ branch\text{-}name,\ balance * 1.06}(\sigma_{balance > 10000}(account)) \\ \cup\ & \Pi_{account\text{-}number,\ branch\text{-}name,\ balance * 1.05}(\sigma_{balance \leq 10000}(account)) \end{aligned}$$

Views

- In some cases, it is not desirable for all users to see the entire logical model i.e. all the actual relations stored in the database.
- Consider "a person who needs to know a customer's loan number but has no need to see the amount." the person should see a relation described, in the relational algebra, by

$$\Pi_{customer\text{-}name,\ loan\text{-}number}\ (borrower \bowtie loan)$$

- Any relation that is made visible to a user as a "virtual Relation" is called a VIEW
  - It provides *Limited access to DB*
  - It presents the *Tailored schema*.

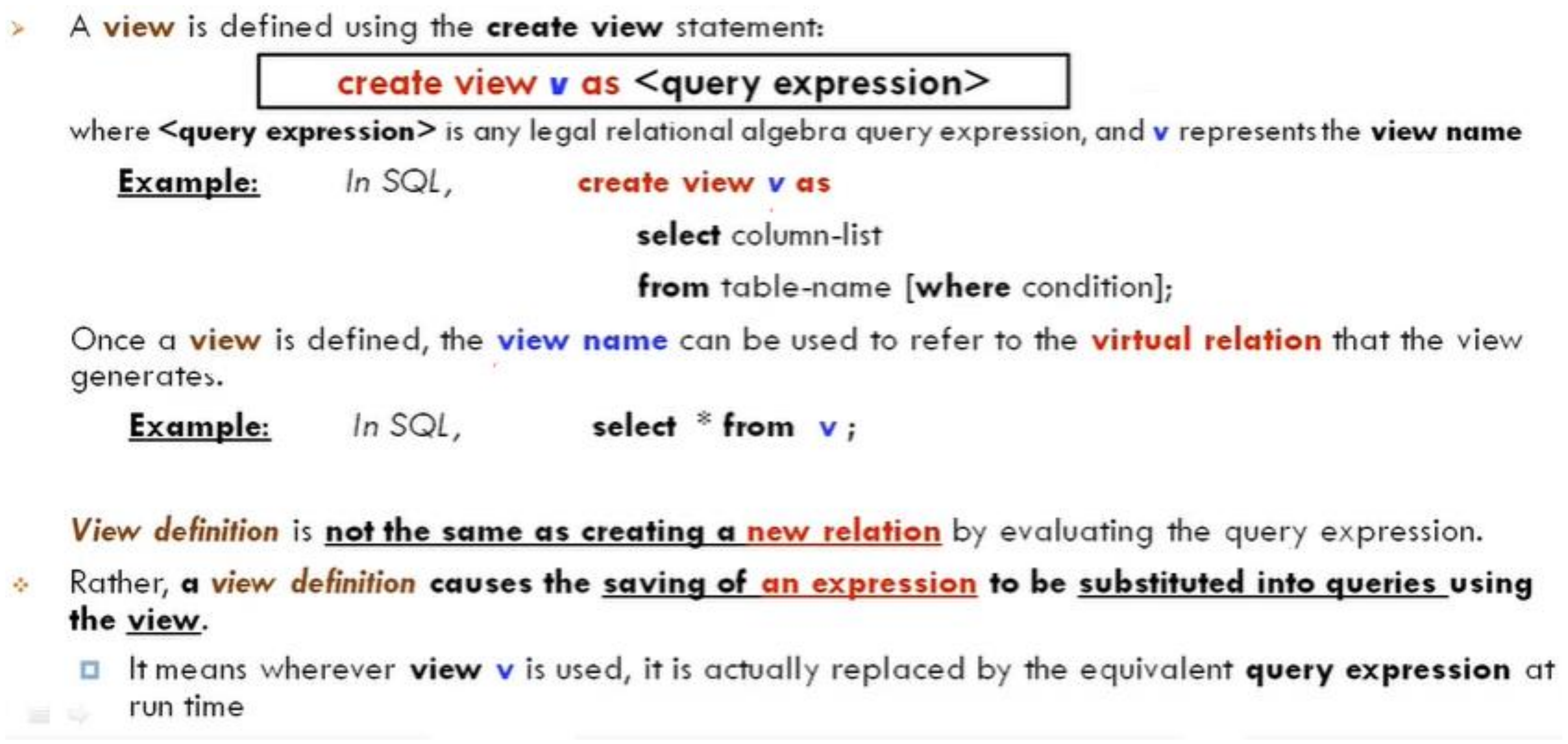
Views

- “Views” are Virtual Relations or Virtual tables, through which a selective portion of the data from one or more relations or tables can be seen.
- Views do not contain data of their own.
- Views do not exist physically.
- Uses of View:
  - ✓ It helps in query processing like simplify commands for the user, store complex queries, etc.
  - ✓ To restrict access to the database
  - ✓ To hide data complexity

Views

- Data modifications like insert, update, delete operations on views affects the actual relations in the database, upon which view is based. (Same in SQL)
- Views are stored in data dictionary in the table called USER_VIEWS.

Views



- A **view** is defined using the **create view** statement:

  $$\boxed{\textbf{create view } v \textbf{ as } \langle\textbf{query expression}\rangle}$$

  where **<query expression>** is any legal relational algebra query expression, and **v** represents the **view name**

  Example: *In SQL,*

  ```
  create view v as
      select column-list
      from table-name [where condition];
  ```

  Once a **view** is defined, the **view name** can be used to refer to the **virtual relation** that the view generates.

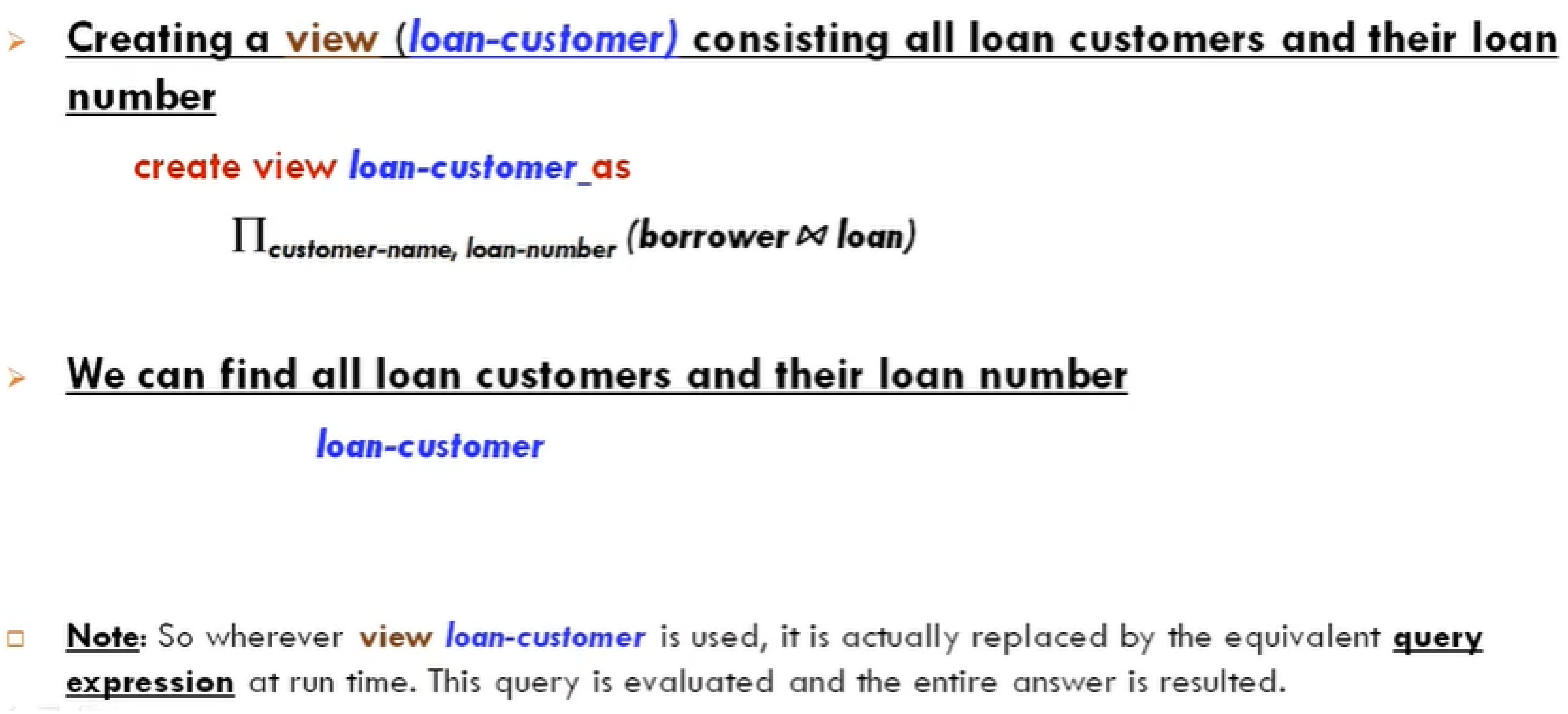
  Example: *In SQL,*

  ```
  select * from v ;
  ```

  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 to be substituted into queries using the view.**
  - It means wherever **view v** is used, it is actually replaced by the equivalent **query expression** at run time

Examples of Views



- **Creating a view (*loan-customer*) consisting all loan customers and their loan number**

  create view *loan-customer*_as

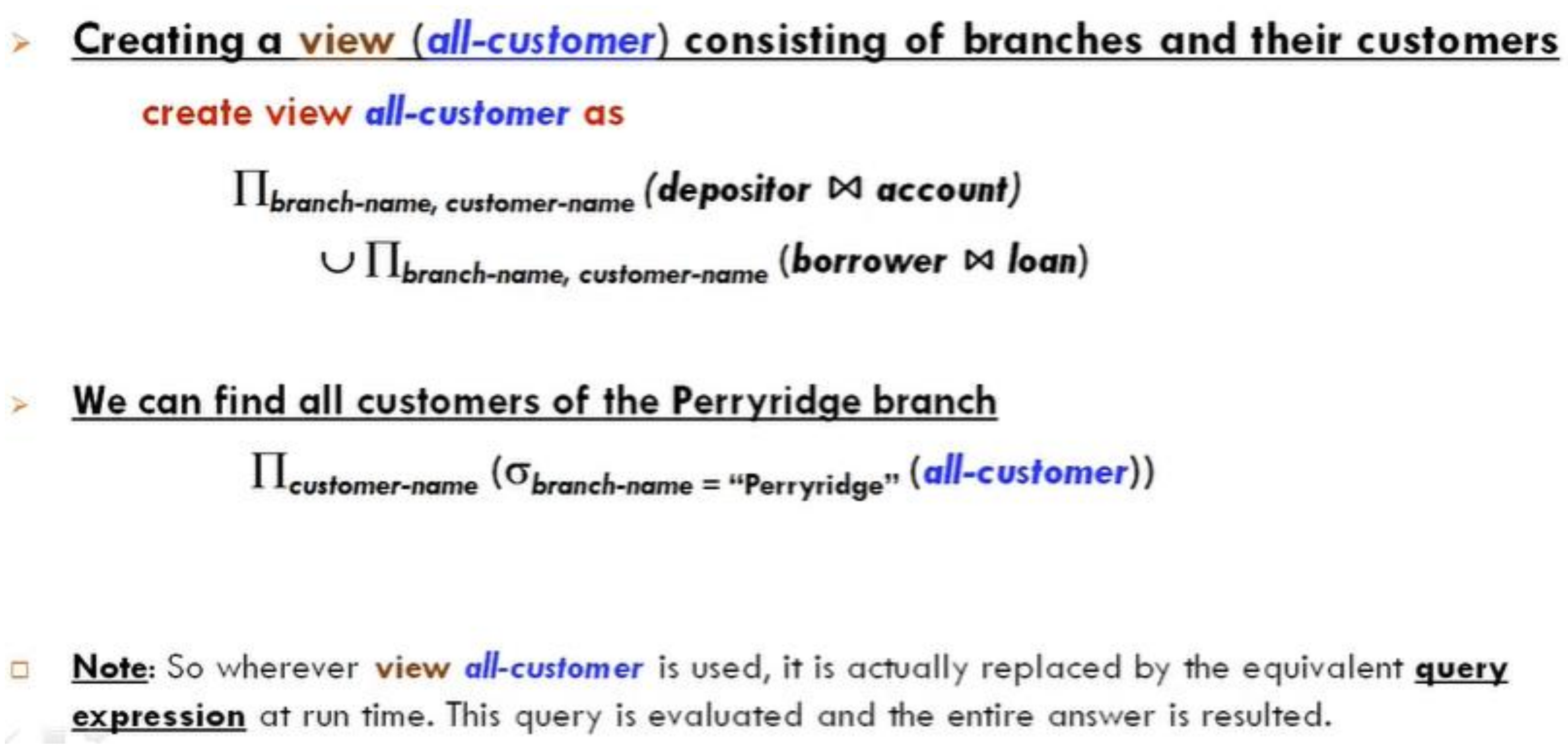
  $$\Pi_{customer\text{-}name,\ loan\text{-}number}\ (borrower \bowtie loan)$$

- **We can find all loan customers and their loan number**

  loan-customer

- **Note**: So wherever view *loan-customer* is used, it is actually replaced by the equivalent **query expression** at run time. This query is evaluated and the entire answer is resulted.

Examples of Views



- **Creating a view (*all-customer*) consisting of branches and their customers**

  create view *all-customer* as

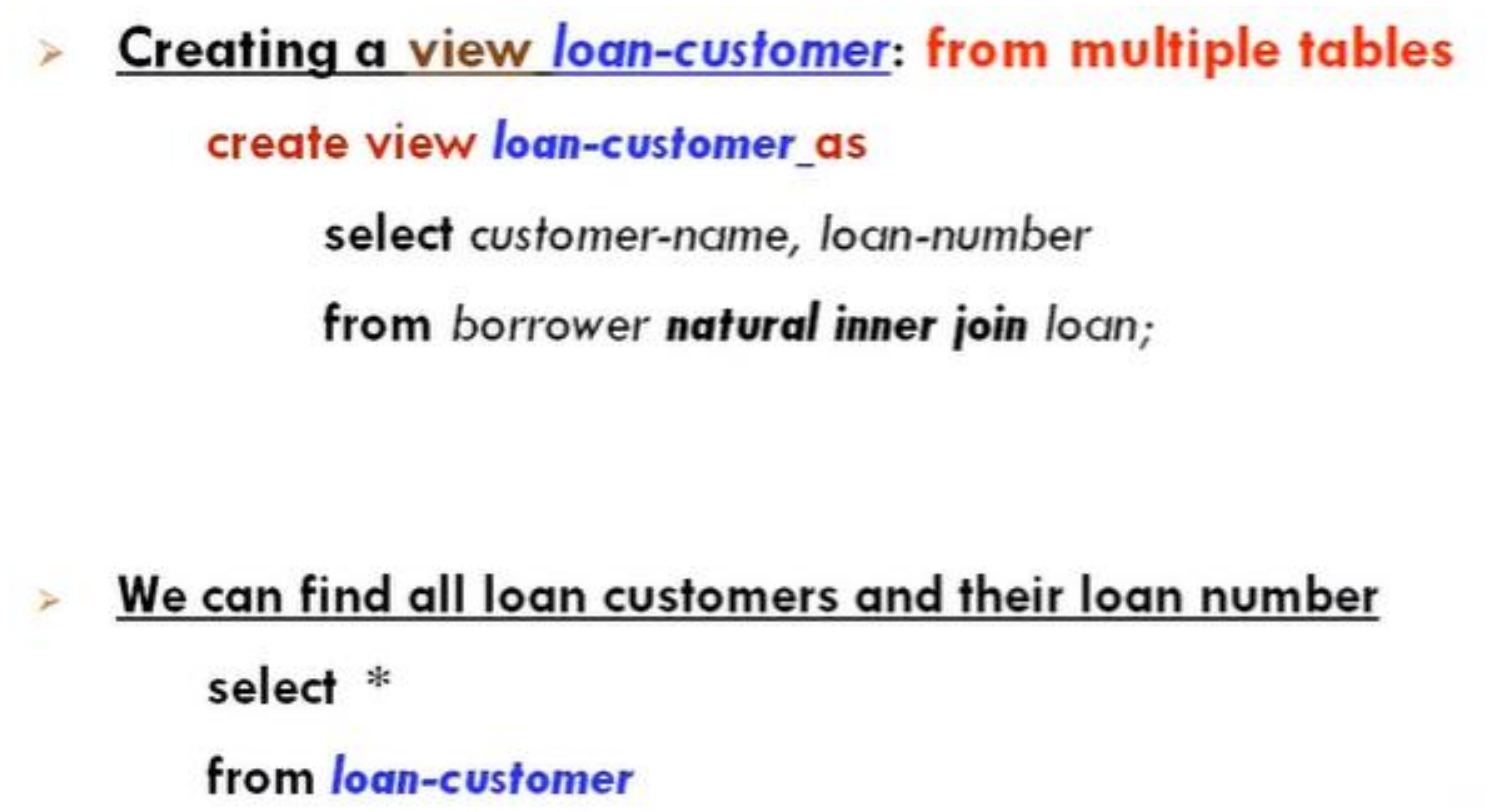
$$\Pi_{\textit{branch-name, customer-name}}\,(\textit{depositor} \bowtie \textit{account})$$
$$\cup\ \Pi_{\textit{branch-name, customer-name}}\,(\textit{borrower} \bowtie \textit{loan})$$

- **We can find all customers of the Perryridge branch**

$$\Pi_{\textit{customer-name}}\,(\sigma_{\textit{branch-name} = \text{“Perryridge”}}\,(\textit{all-customer}))$$

- **Note**: So wherever view *all-customer* is used, it is actually replaced by the equivalent **query expression** at run time. This query is evaluated and the entire answer is resulted.

Examples of Views in SQL



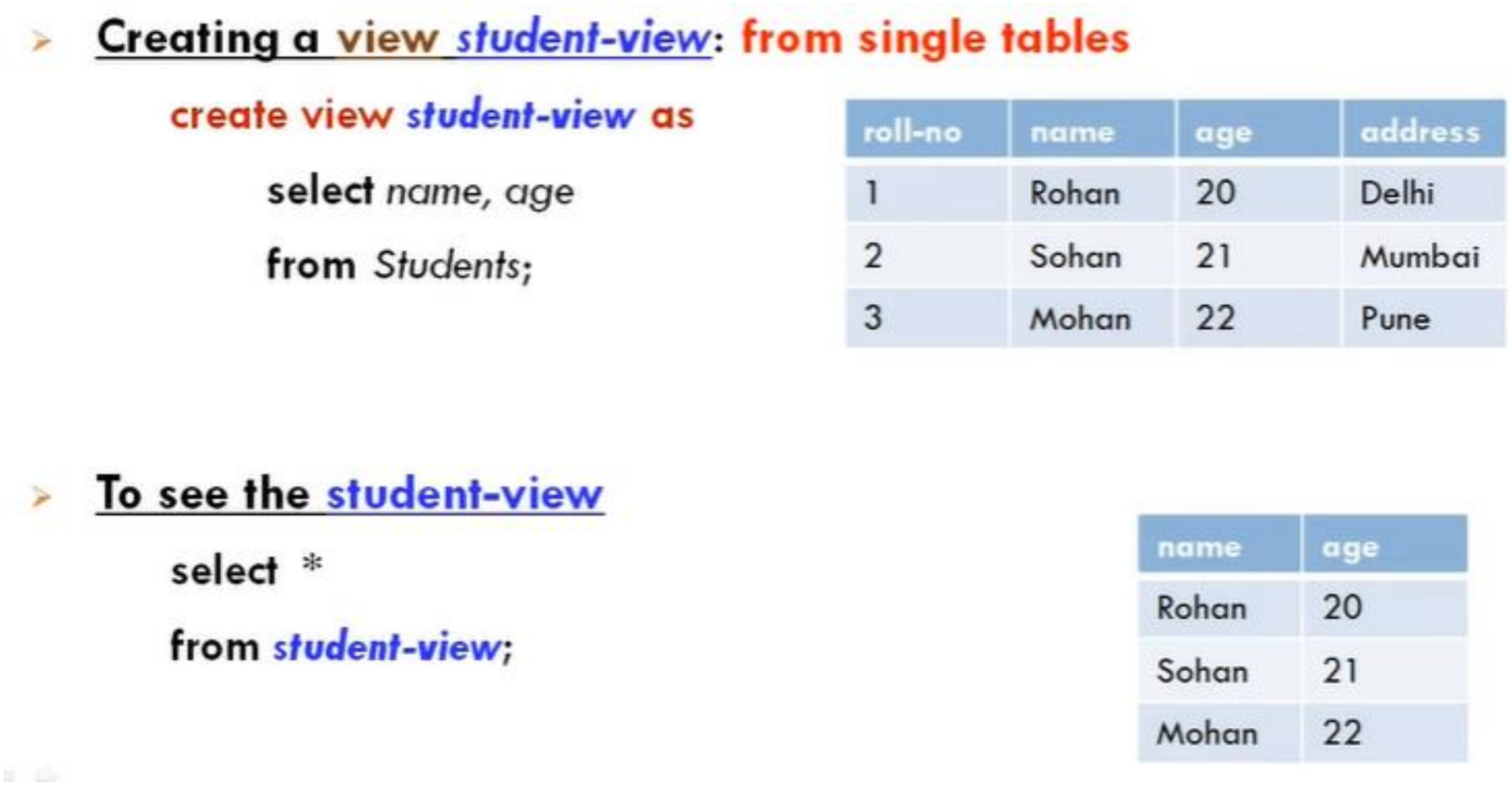
- **Creating a view *loan-customer*: from multiple tables**

  ```
  create view loan-customer_as
          select customer-name, loan-number
          from borrower natural inner join loan;
  ```

- **We can find all loan customers and their loan number**

  ```
  select  *
  from loan-customer
  ```

Examples of Views in SQL



- **Creating a view *student-view*: from single tables**

```
create view student-view as
        select name, age
        from Students;
```

roll-no	name	age	address
1	Rohan	20	Delhi
2	Sohan	21	Mumbai
3	Mohan	22	Pune

- **To see the student-view**

```
select *
from student-view;
```

name	age
Rohan	20
Sohan	21
Mohan	22

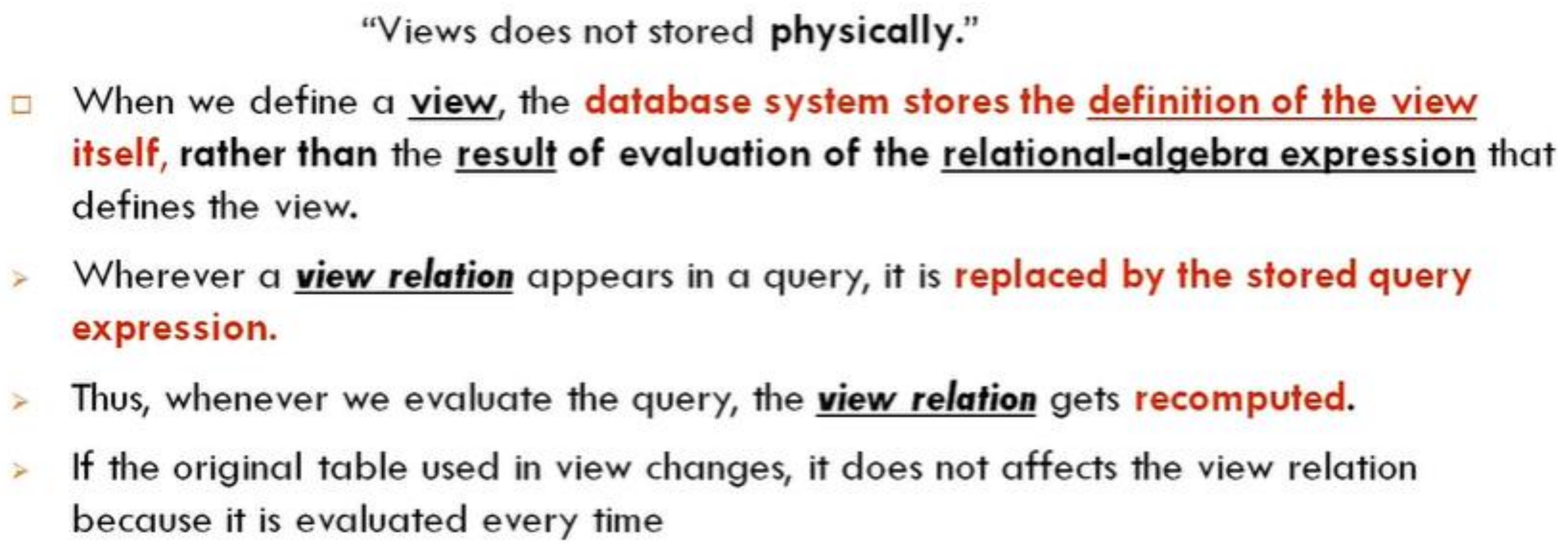
Drop View

- A view can be deleted using the **Drop View** statement.

```
drop view viewname
```

- Example: **drop view** ***student-view***

Views : Summary



"Views does not stored **physically.**"

- When we define a <u>**view**</u>, the **database system stores the <u>definition of the view</u> itself, rather than** the **<u>result</u> of evaluation of the <u>relational-algebra expression</u>** that defines the view.

- Wherever a ***<u>view relation</u>*** appears in a query, it is **replaced by the stored query expression.**

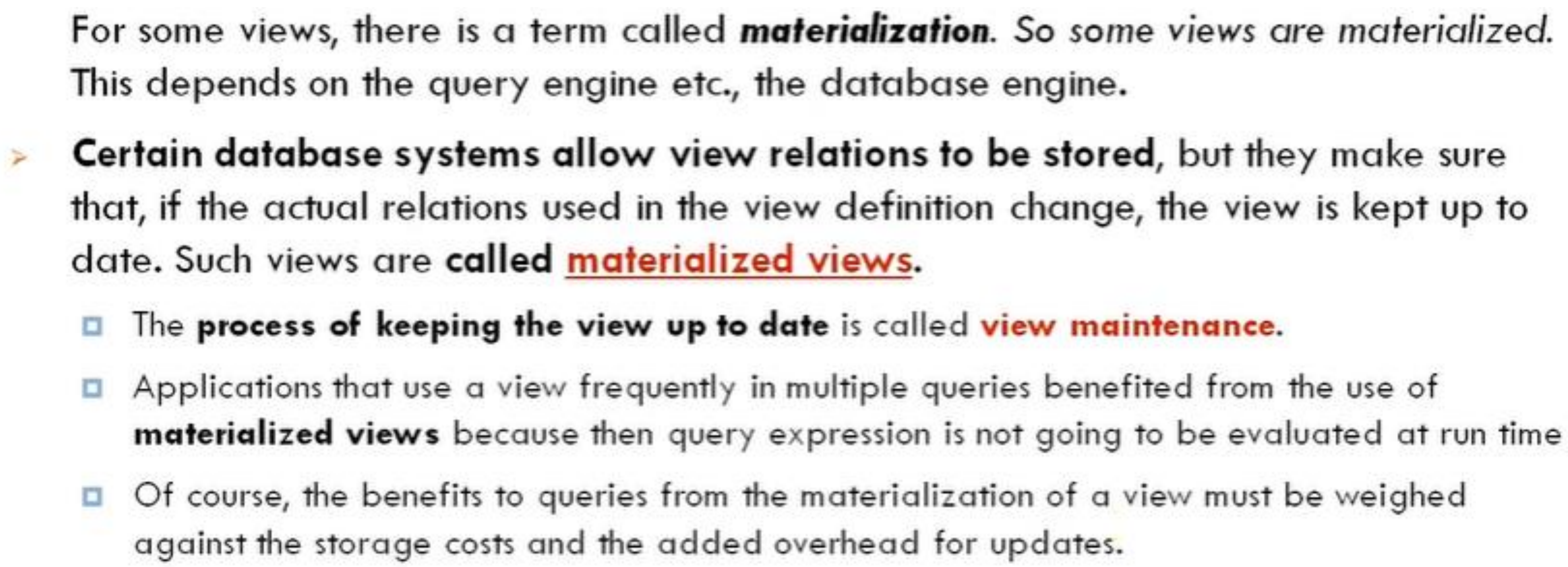
- Thus, whenever we evaluate the query, the ***<u>view relation</u>*** gets **recomputed.**

- If the original table used in view changes, it does not affects the view relation because it is evaluated every time

Types of Views

- ***Read-only View :***
  - Used only to read data.
  - In SQL, it allows only SELECT operations.
- ***Updateable View :***
  - Used to read and update the data.
  - In SQL, it allows SELECT as well as INSERT , UPDATE and DELETE operations.

Materialized Views



For some views, there is a term called ***materialization***. *So some views are materialized.* This depends on the query engine etc., the database engine.

- **Certain database systems allow view relations to be stored**, but they make sure that, if the actual relations used in the view definition change, the view is kept up to date. Such views are **called** materialized views**.**
  - The **process of keeping the view up to date** is called **view maintenance.**
  - Applications that use a view frequently in multiple queries benefited from the use of **materialized views** because then query expression is not going to be evaluated at run time
  - Of course, the benefits to queries from the materialization of a view must be weighed against the storage costs and the added overhead for updates.