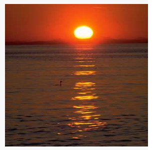
Chapter 14: Query Optimization

Database System Concepts 5th Ed.



Chapter 14: Query Optimization

- Introduction
- Transformation of Relational Expressions
- Equivalence Rules
- Example of Transformation

Introduction

- **Query optimization** is the process of selecting the most efficient query-evaluation plan from among the various strategies that are possible for processing a given query.
- The database system is required to construct a query-evaluation plan that minimizes the cost of query evaluation.
- Consider the relational-algebra expression for the query: **"Find the names of all customers who have an account at any branch located in Brooklyn."**

$$\Pi_{customer\text{-}name}\ (\sigma_{branch-city\,=\,\text{"Brooklyn"}}\ (branch \bowtie\ (account \bowtie\ depositor)))$$

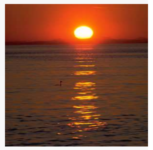
- This expression constructs a large intermediate relation due to the join operation of three relations: **branch ⋈ account ⋈ depositor**. However, the response to the above query requires only a few tuples of this relation.

Introduction

- Since we are concerned with only those tuples in the *branch* relation that pertain to branches located in Brooklyn, we do not need to consider those tuples that do not have *branch-city* = "Brooklyn".
- By reducing the number of tuples of the *branch* relation that we need to access, we can reduce the size of the intermediate result.
- Our query is now represented by the following relational-algebra expression:

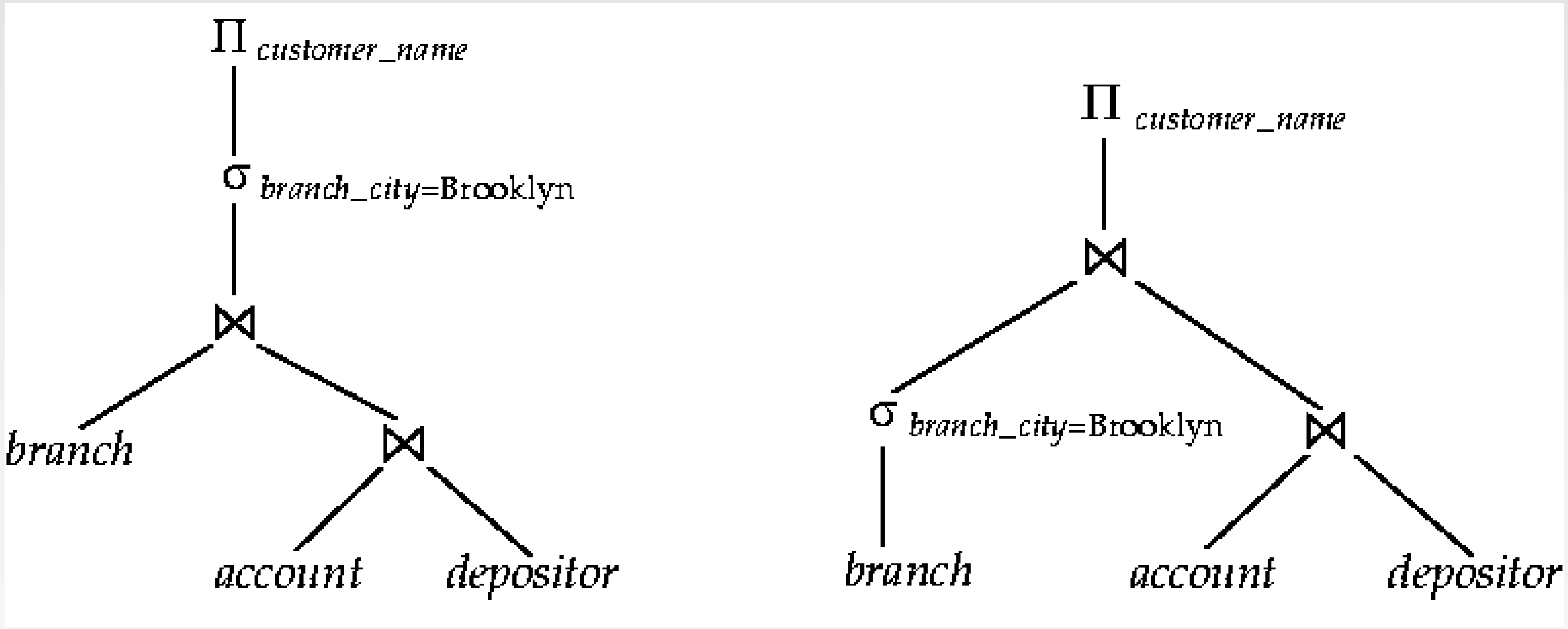
$$\Pi_{customer\text{-}name}\ ((\sigma_{branch\text{-}city\,=\,\text{"Brooklyn"}}\ (branch)) \bowtie\ (account \bowtie\ depositor))$$

which is equivalent to the original relational algebra expression, but it generates smaller intermediate relations.



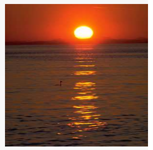
Introduction

- The following figure depicts the initial and transformed expressions:



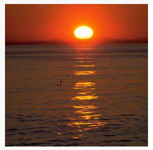
Transformation of Relational Algebra Expressions

- Two Relational Algebra (RA) expressions are said to be equivalent if on every legal database instance, the two expressions generate the same set of tuples.
- A **legal database instance** refers to that database system which satisfies all the integrity constraints specified in the database schema.
- The query optimizer uses equivalence rules to transform RA expressions into other equivalent RA expressions.
- **Equivalence Rule:** An equivalence rule specifies how to transform a RA expression into a logically equivalent expression such that the original expression and the transformed expression are equivalent.
- We can replace an expression of the first form by an expression of the second form or vise versa because both expressions produce the same output on any legal database instance.

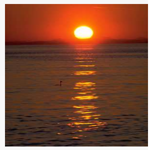


Transformation of Relational Algebra Expressions

- Given a RA expression, it is the job of the query optimizer to come up with a query-evaluation plan that computes the same result as the given expression, and is the least costly way of generating the result.
- An **evaluation plan** defines exactly what algorithm is used for each operation, and how the execution of the operations is coordinated.
- To choose among different query-evaluation plans, the optimizer has to estimate the cost of each evaluation plan.
- Steps in **cost-based query optimization:**
  1. Generate logically equivalent expressions using **equivalence rules**
  2. Annotate resultant expressions to get alternative query evaluation plans
  3. Choose the cheapest plan based on its **estimated cost**

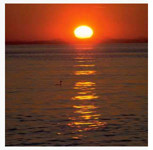


Generating Equivalent Expressions



Equivalence Rules

- An **equivalence rule** says that expressions of two forms are equivalent.
- We can replace an expression of the first form by an expression of the second form, or vice versa. Thus, we can replace an expression of the second form by an expression of the first form since the two expressions would generate the same result on any valid database.
- The optimizer uses equivalence rules to transform expressions into other logically equivalent expressions.
- **Notations:**
  - θ_1, θ_2, θ_3 and so on denote **predicates** (selection condition using attributes and operation)
  - L_1, L_2, L_3, and so on to denote lists of **attributes**
  - E_1,E_2, E_3 and so on to denote **relational-algebra expressions**.
  - *r* is a **relation name**



Equivalence Rules

1. Conjunctive selection operations can be decomposed into a sequence of individual selections.

$$\sigma_{\theta_1 \wedge \theta_2}(E) = \sigma_{\theta_1}(\sigma_{\theta_2}(E))$$

2. Selection operations are commutative.

$$\sigma_{\theta_1}(\sigma_{\theta_2}(E)) = \sigma_{\theta_2}(\sigma_{\theta_1}(E))$$

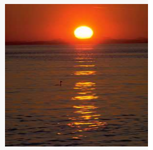
3. Only the final operation in a sequence of projection operations is needed, the rest can be omitted.

$$\Pi_{L_1}(\Pi_{L_2}(\ldots(\Pi_{Ln}(E))\ldots)) = \Pi_{L_1}(E)$$

4. Selection operation can be combined with Cartesian products and theta joins in the following manner:

$$\sigma_{\theta}(E_1 \times E_2) = E_1 \bowtie_{\theta} E_2$$

$$\sigma_{\theta 1}(E_1 \bowtie_{\theta 2} E_2) = E_1 \bowtie_{\theta 1 \wedge\, \theta 2} E_2$$



Equivalence Rules (Cont.)

5. Theta-join operations (and natural joins) are commutative.

$$E_1 \bowtie_\theta E_2 = E_2 \bowtie_\theta E_1$$

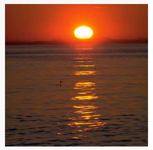
6. (a) Natural join operations are associative in the following manner:

$$(E_1 \bowtie E_2) \bowtie E_3 = E_1 \bowtie (E_2 \bowtie E_3)$$

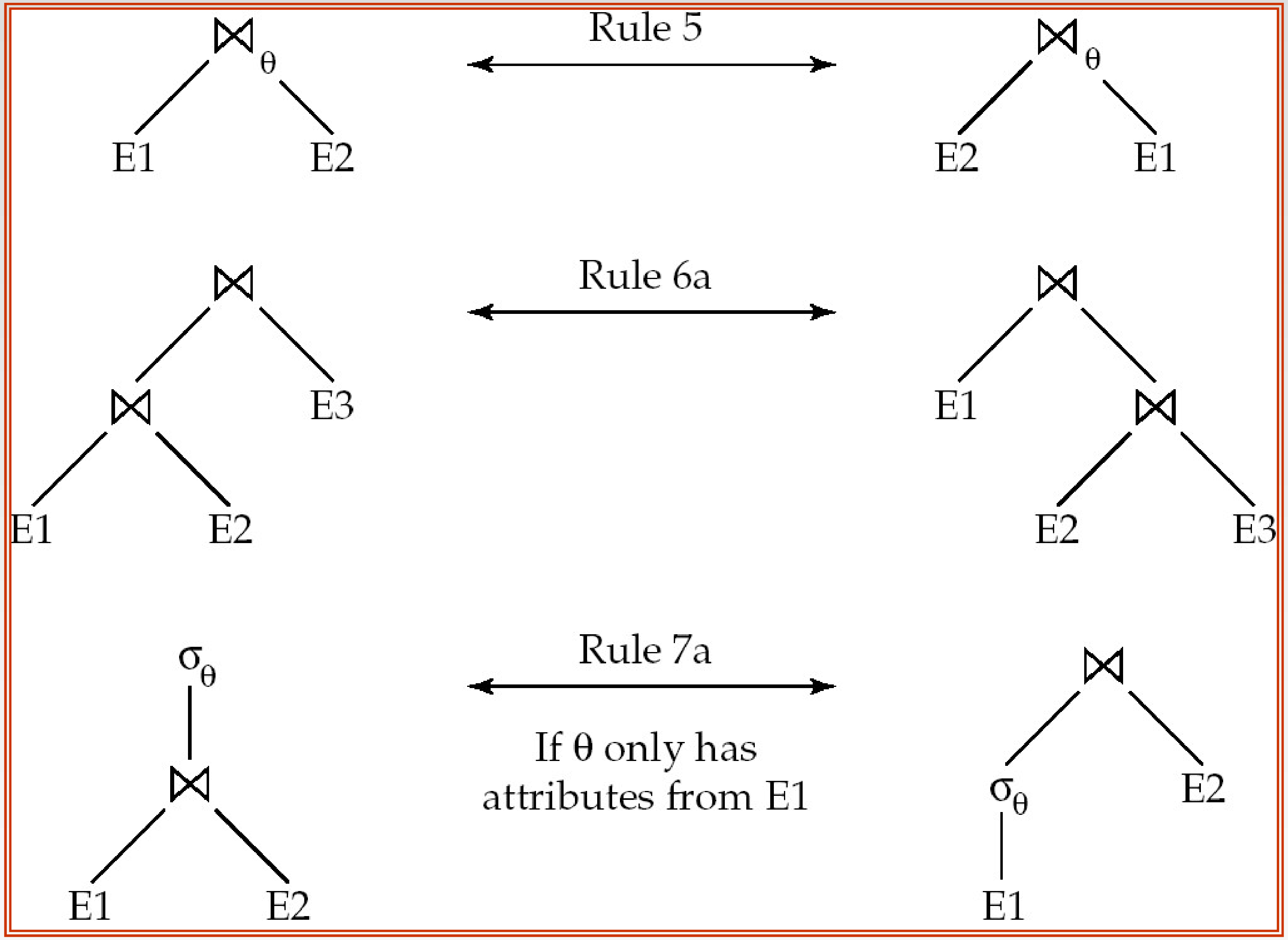
(b) Theta joins are associative in the following manner:

$$(E_1 \bowtie_{\theta 1} E_2) \bowtie_{\theta 2 \wedge \theta 3} E_3 = E_1 \bowtie_{\theta 1 \wedge \theta 3} (E_2 \bowtie_{\theta 2} E_3)$$

where, θ_2 involves attributes only from E_2 and E_3.



Pictorial Depiction of Equivalence Rules



Equivalence Rules (Cont.)

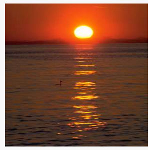
7. The selection operation distributes over the theta join operation under the following two conditions:

   (a) When all the attributes in θ involve only the attributes of one of the expressions (E_1) being joined.

$$\sigma_{\theta}(\mathbf{E_1} \bowtie_{\theta 0} \mathbf{E_2}) = (\sigma_{\theta}(\mathbf{E_1})) \bowtie_{\theta 0} \mathbf{E_2}$$

   (b) When θ_1 involves only the attributes of E_1 and θ_2 involves only the attributes of E_2.

$$\sigma_{\theta 1 \wedge \theta 2}(\mathbf{E_1} \bowtie_{\theta} \mathbf{E_2}) = (\sigma_{\theta 1}(\mathbf{E_1})) \bowtie_{\theta} (\sigma_{\theta 2}(\mathbf{E_2}))$$



Equivalence Rules (Cont.)

8. The projection operation distributes over the theta join operation under the following conditions:

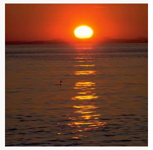
   (a) Let $L1$ and $L2$ be attributes of $E1$ and $E2$ respectively. If θ involves only attributes from $L_1 \cup L_2$:

   $$\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = (\Pi_{L_1}(E_1)) \bowtie_\theta (\Pi_{L_2}(E_2))$$

   (b) Consider a join $E_1 \bowtie_\theta E_2$.

   - Let L_1 and L_2 be sets of attributes from E_1 and E_2, respectively.
   - Let L_3 be attributes of E_1 that are involved in join condition θ, but are not in $L_1 \cup L_2$, and
   - Let L_4 be attributes of E_2 that are involved in join condition θ, but are not in $L_1 \cup L_2$.

   $$\Pi_{L_1 \cup L_2}(E_1 \bowtie_\theta E_2) = \Pi_{L_1 \cup L_2}((\Pi_{L_1 \cup L_3}(E_1)) \bowtie_\theta (\Pi_{L_2 \cup L_4}(E_2)))$$



Equivalence Rules (Cont.)

9. The set operations union and intersection are commutative

$$E_1 \cup E_2 = E_2 \cup E_1$$
$$E_1 \cap E_2 = E_2 \cap E_1$$

   - Set difference is not commutative.

10. Set union and intersection are associative.

$$(E_1 \cup E_2) \cup E_3 = E_1 \cup (E_2 \cup E_3)$$
$$(E_1 \cap E_2) \cap E_3 = E_1 \cap (E_2 \cap E_3)$$

11. The selection operation distributes over $\cup$, $\cap$ and $-$.

$$\sigma_\theta (E_1 - E_2) = \sigma_\theta (E_1) - \sigma_\theta(E_2)$$

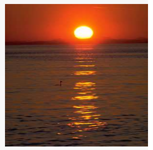
and similarly, for $\cup$ and $\cap$ in place of $-$

Also: $$\sigma_\theta (E_1 - E_2) = \sigma_\theta(E_1) - E_2$$

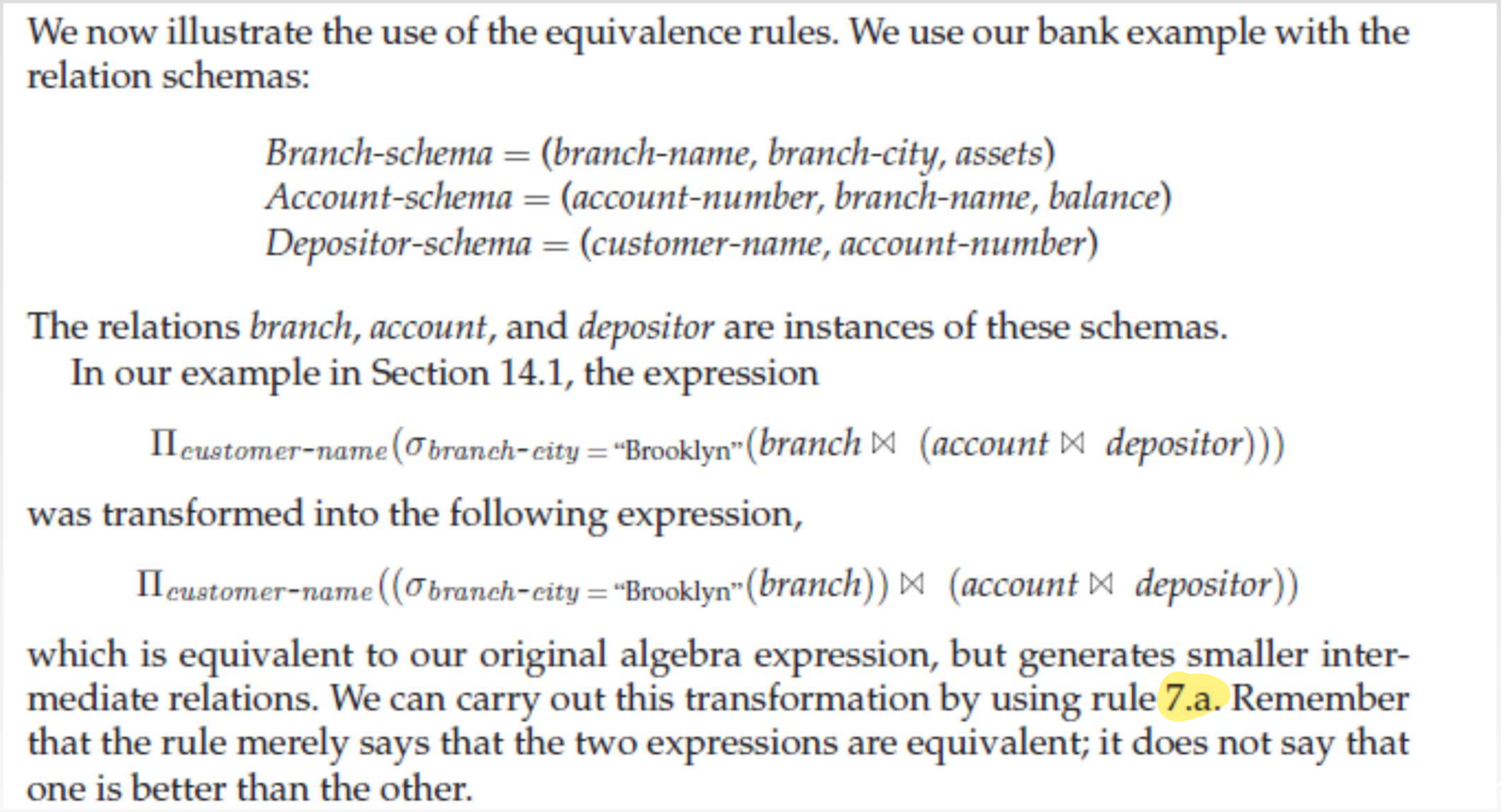
and similarly, for $\cap$ in place of $-$, but not for $\cup$

12. The projection operation distributes over union

$$\Pi_L(E_1 \cup E_2) = (\Pi_L(E_1)) \cup (\Pi_L(E_2))$$



Example of Transformations



We now illustrate the use of the equivalence rules. We use our bank example with the relation schemas:

$$\begin{aligned}&\textit{Branch-schema} = (\textit{branch-name}, \textit{branch-city}, \textit{assets})\\&\textit{Account-schema} = (\textit{account-number}, \textit{branch-name}, \textit{balance})\\&\textit{Depositor-schema} = (\textit{customer-name}, \textit{account-number})\end{aligned}$$

The relations *branch*, *account*, and *depositor* are instances of these schemas.

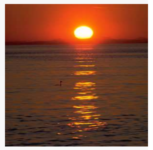
In our example in Section 14.1, the expression

$$\Pi_{customer\text{-}name}(\sigma_{branch\text{-}city = \text{"Brooklyn"}}(branch \bowtie (account \bowtie depositor)))$$

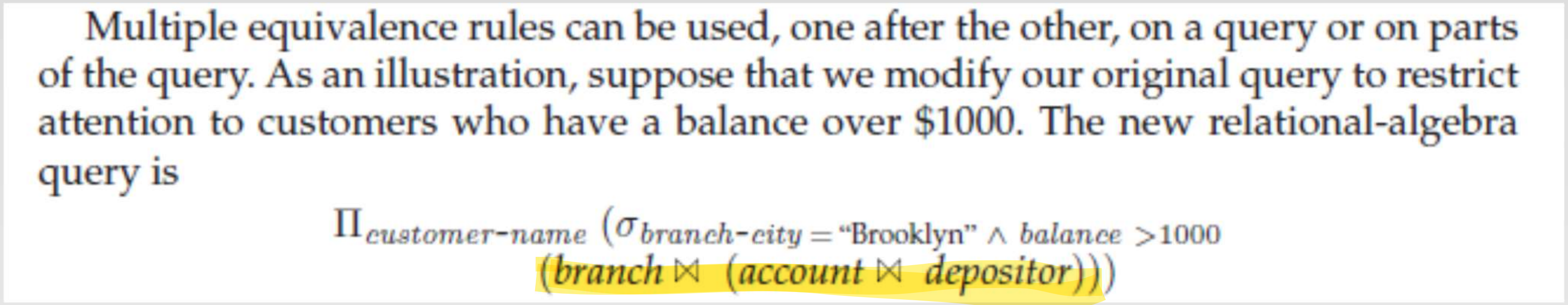
was transformed into the following expression,

$$\Pi_{customer\text{-}name}((\sigma_{branch\text{-}city = \text{"Brooklyn"}}(branch)) \bowtie (account \bowtie depositor))$$

which is equivalent to our original algebra expression, but generates smaller intermediate relations. We can carry out this transformation by using rule 7.a. Remember that the rule merely says that the two expressions are equivalent; it does not say that one is better than the other.

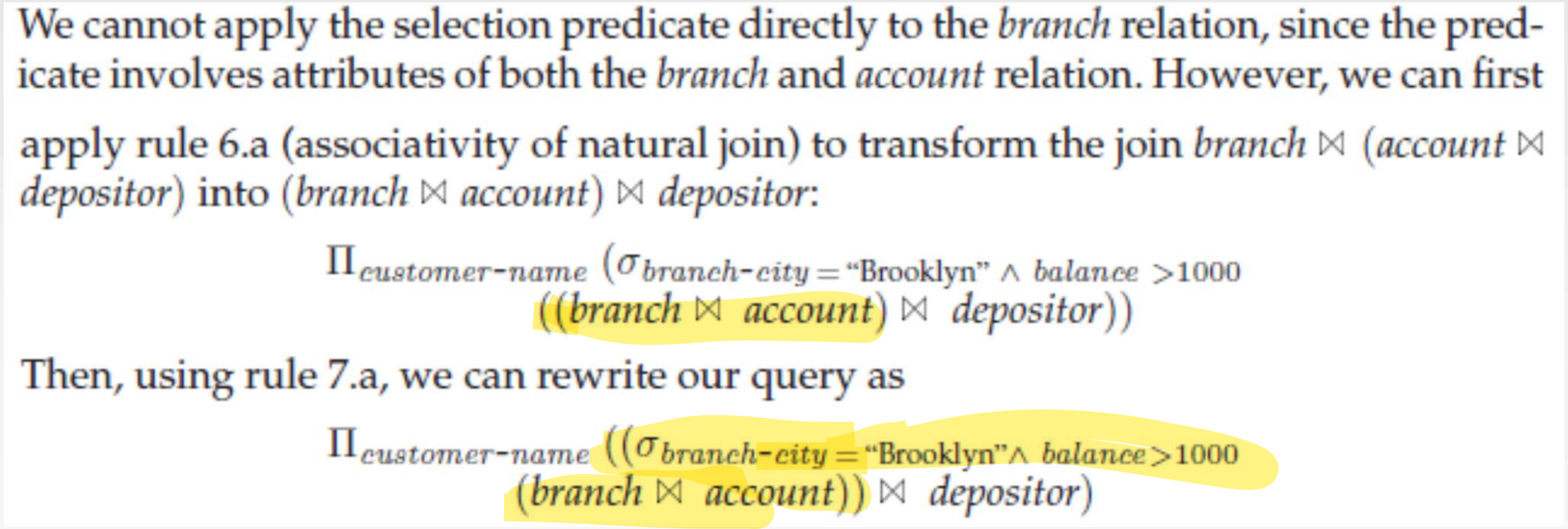


Example of Transformations



Multiple equivalence rules can be used, one after the other, on a query or on parts of the query. As an illustration, suppose that we modify our original query to restrict attention to customers who have a balance over $1000. The new relational-algebra query is

$$\Pi_{customer\text{-}name}\ (\sigma_{branch\text{-}city = \text{"Brooklyn"}\ \wedge\ balance\ > 1000}\ (branch \bowtie\ (account \bowtie\ depositor)))$$



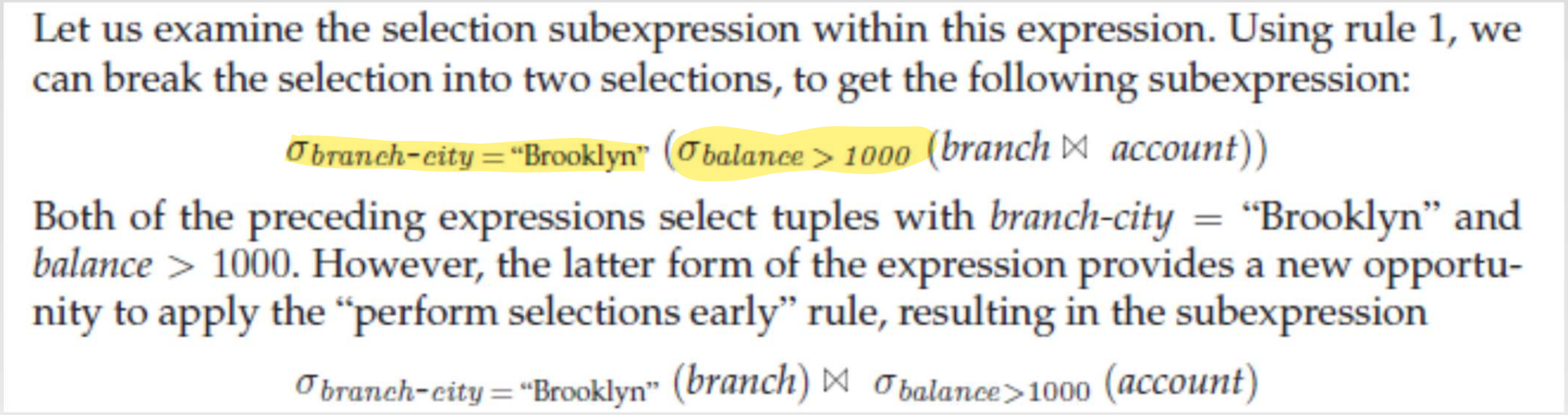
We cannot apply the selection predicate directly to the *branch* relation, since the predicate involves attributes of both the *branch* and *account* relation. However, we can first apply rule 6.a (associativity of natural join) to transform the join *branch* ⋈ (*account* ⋈ *depositor*) into (*branch* ⋈ *account*) ⋈ *depositor*:

$$\Pi_{customer\text{-}name}\ (\sigma_{branch\text{-}city = \text{"Brooklyn"}\ \wedge\ balance\ > 1000}\ ((branch \bowtie\ account) \bowtie\ depositor))$$

Then, using rule 7.a, we can rewrite our query as

$$\Pi_{customer\text{-}name}\ ((\sigma_{branch\text{-}city = \text{"Brooklyn"} \wedge\ balance > 1000}\ (branch \bowtie\ account)) \bowtie\ depositor)$$

Example of Transformations



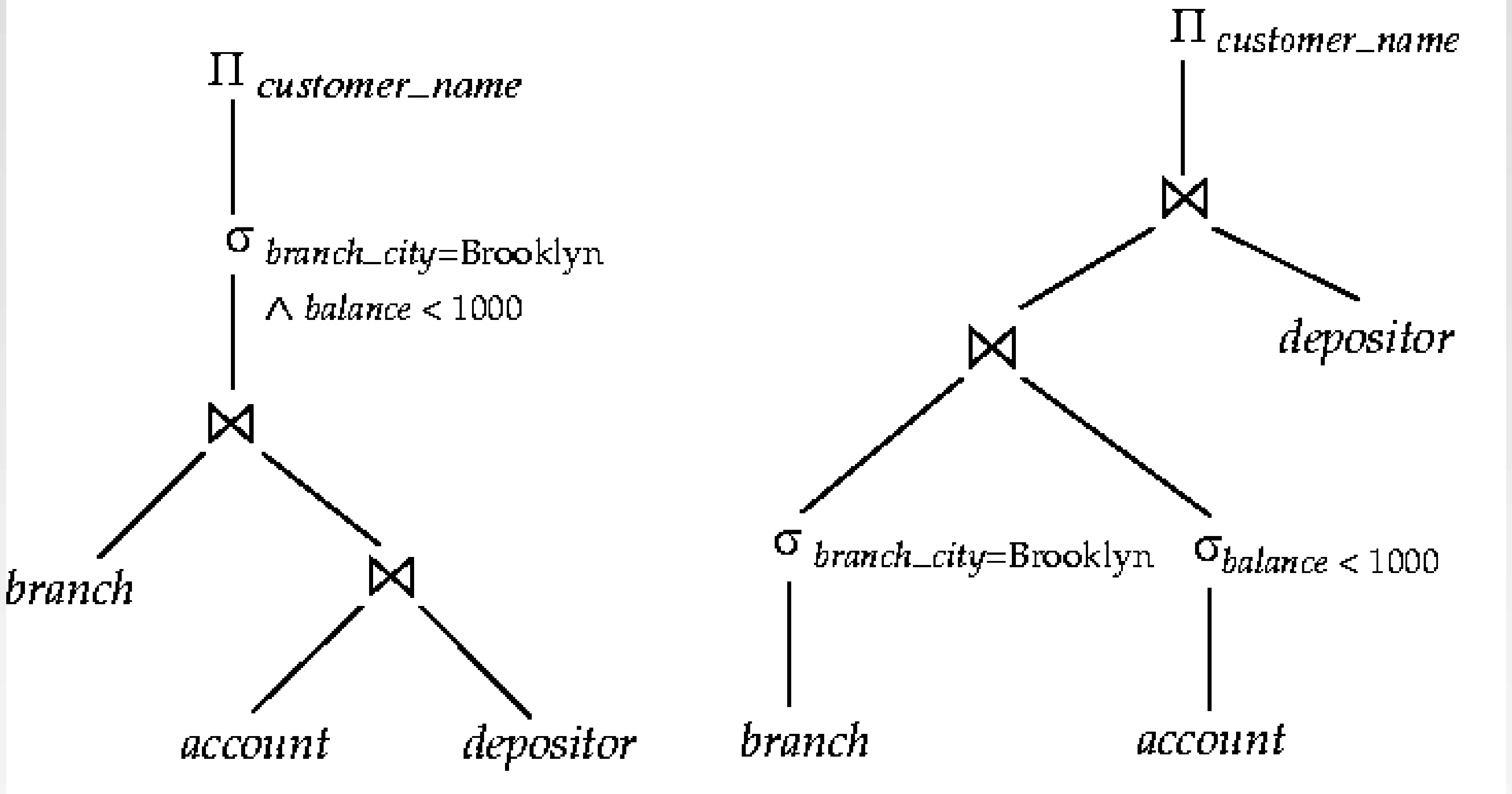
Let us examine the selection subexpression within this expression. Using rule 1, we can break the selection into two selections, to get the following subexpression:

$$\sigma_{branch\text{-}city = \text{“Brooklyn”}} \left(\sigma_{balance > 1000} \left(branch \bowtie account\right)\right)$$

Both of the preceding expressions select tuples with *branch-city* = “Brooklyn” and *balance* > 1000. However, the latter form of the expression provides a new opportunity to apply the “perform selections early” rule, resulting in the subexpression

$$\sigma_{branch\text{-}city = \text{“Brooklyn”}} (branch) \bowtie \sigma_{balance > 1000} (account)$$

Tree after Multiple Transformations



$\Pi_{customer\_name}$
$\sigma_{branch\_city=\text{Brooklyn} \wedge balance < 1000}$
$\bowtie$
branch
$\bowtie$
account
depositor
$\Pi_{customer\_name}$
$\bowtie$
$\bowtie$
depositor
$\sigma_{branch\_city=\text{Brooklyn}}$
$\sigma_{balance < 1000}$
branch
account

(a) Initial expression tree (b) Tree after multiple transformations