Material for Assignment 1

This material is taken from the 5th edition of Database System Concepts by Silberschatz, Korth, and Sudarshan.

This document contains the following details:

- Pages 2-3: Book problems for Assignment 1
  Only problems with an arrow "→" next to them need to be completed.
- Page 4: Database schema for problems 2.5, 2.7 and 2.9 (at bottom of page)
- Page 5: The banking database schema for problems 2.6 and 2.8, along with additional details for these problems
- Pages 6-7: The book description of the relational division operator ÷
- Pages 8-11: The book material on equivalence rules for relational algebra expressions

2.2 The outer-join operations extend the natural-join operation so that tuples from the participating relations are not lost in the result of the join. Describe how the theta join operation can be extended so that tuples from the left, right, or both relations are not lost from the result of a theta join.

2.3 Consider the relational database of Figure 2.35. Give an expression in the relational algebra for each request:

a. Modify the database so that Jones now lives in Newtown.
b. Give all managers in this database a 10 percent salary raise.

Exercises

2.4 Describe the differences in meaning between the terms *relation* and *relation schema*.

2.5 Consider the relational database of Figure 2.35, where the primary keys are underlined. Give an expression in the relational algebra to express each of the following queries:

a. Find the names of all employees who work for First Bank Corporation.
b. Find the names and cities of residence of all employees who work for First Bank Corporation.
c. Find the names, street address, and cities of residence of all employees who work for First Bank Corporation and earn more than $10,000 per annum.
d. Find the names of all employees in this database who live in the same city as the company for which they work.
e. Assume the companies may be located in several cities. Find all companies located in every city in which Small Bank Corporation is located.

2.6 Consider the relation of Figure 2.20, which shows the result of the query "Find the names of all customers who have a loan at the bank." Rewrite the query to include not only the name, but also the city of residence for each customer. Observe that now customer Jackson no longer appears in the result, even though Jackson does in fact have a loan from the bank.

a. Explain why Jackson does not appear in the result.
b. Suppose that you want Jackson to appear in the result. How would you modify the database to achieve this effect?
c. Again, suppose that you want Jackson to appear in the result. Write a query using an outer join that accomplishes this desire without your having to modify the database.

2.7 Consider the relational database of Figure 2.35. Give an expression in the relational algebra for each request:

a. Give all employees of First Bank Corporation a 10 percent salary raise.
b. Give all managers in this database a 10 percent salary raise, unless the salary would be greater than $100,000. In such cases, give only a 3 percent raise.
c. Delete all tuples in the *works* relation for employees of Small Bank Corporation.

2.8 Using the bank example, write relational-algebra queries to find the accounts held by more than two customers in the following ways:

a. Using an aggregate function.
b. Without using any aggregate functions.

2.9 Consider the relational database of Figure 2.35. Give a relational-algebra expression for each of the following queries:

a. Find the company with the most employees.
b. Find the company with the smallest payroll.
c. Find those companies whose employees earn a higher salary, on average, than the average salary at First Bank Corporation.

2.10 List two reasons why null values might be introduced into the database.

2.11 Consider the following relational schema

employee(*empno*, *name*, *office*, *age*)
books(*isbn*, *title*, *authors*, *publisher*)
loan(*empno*, *isbn*, *date*)

Write the following queries in relational algebra.

a. Find the names of employees who have borrowed a book published by McGraw-Hill.
b. Find the names of employees who have borrowed all books published by McGraw-Hill.
c. Find the names of employees who have borrowed more than five different books published by McGraw-Hill.
d. For each publisher, find the names of employees who have borrowed more than five books of that publisher.

Bibliographical Notes

E. F. Codd of the IBM San Jose Research Laboratory proposed the relational model in the late 1960s (Codd [1970]). This work led to the prestigious ACM Turing Award to Codd in 1981 (Codd [1982]).

After Codd published his original paper, several research projects were formed with the goal of constructing practical relational database systems including System R at the IBM San Jose Research Laboratory, Ingres at the University of California at Berkeley, and Query-by-Example at the IBM T. J. Watson Research Center.

Atzeni and Antonellis [1993] and Maier [1983] are texts devoted exclusively to the theory of the relational data model.

The original definition of relational algebra is in Codd [1970]. Extensions to the relational model and discussions of incorporation of null values in the relational algebra (the RM/T model), as well as outer joins, are in Codd [1979]. Codd [1990] is a compendium of E. F. Codd's papers on the relational model. Outer joins are also discussed in Date [1993b].

- Relation schema
- Relation instance
- Keys
- Foreign key
  - ☐ Referencing relation
  - ☐ Referenced relation
- Schema diagram
- Query language
- Procedural language
- Nonprocedural language
- Relational algebra
- Relational-algebra operations
  - ☐ Select σ
  - ☐ Project Π
  - ☐ Union $\cup$
  - ☐ Set difference $-$
  - ☐ Cartesian product $\times$
  - ☐ Rename ρ
- Additional operations
  - ☐ Set intersection $\cap$
  - ☐ Natural join $\bowtie$
  - ☐ Division $\div$
- Assignment operation
- Extended relational-algebra operations
  - ☐ Generalized projection Π
  - ☐ Outer join
    - — Left outer join $⟕$
    - — Right outer join $⟖$
    - — Full outer join $⟗$
  - ☐ Aggregation $\mathcal{G}$
- Multisets
- Grouping
- Null value
- Truth values
  - ☐ *true*
  - ☐ *false*
  - ☐ *unknown*
- Modification of the database
  - ☐ Deletion
  - ☐ Insertion
  - ☐ Updating

Practice Exercises

2.1 Consider the relational database of Figure 2.35, where the primary keys are underlined. Give an expression in the relational algebra to express each of the following queries:

a. Find the names of all employees who live in the same city and on the same street as do their managers.

b. Find the names of all employees in this database who do not work for First Bank Corporation.

c. Find the names of all employees who earn more than every employee of Small Bank Corporation.

employee (*person_name*, *street*, *city*)
works (*person_name*, *company_name*, *salary*)
company (*company_name*, *city*)
manages (*person_name*, *manager_name*)

Figure 2.35 Relational database for Exercises 2.1, 2.3, 2.5, 2.7, and 2.9.

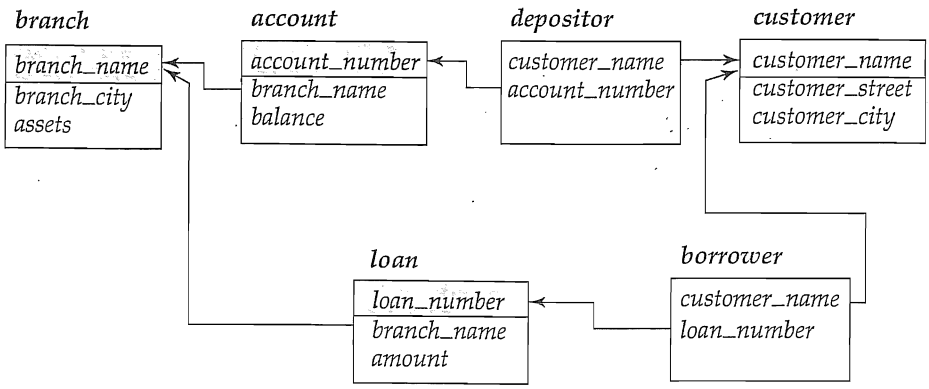

Figure 2.8 Schema diagram for the banking enterprise.

customer_name	*customer_street*	*customer_city*
Adams	Spring	Pittsfield
Brooks	Senator	Brooklyn
Curry	North	Rye
Glenn	Sand Hill	Woodside
Green	Walnut	Stamford
Hayes	Main	Harrison
Johnson	Alma	Palo Alto
Jones	Main	Harrison
Lindsay	Park	Pittsfield
Smith	North	Rye
Turner	Putnam	Stamford
Williams	Nassau	Princeton

Figure 2.4 The *customer* relation.

customer_name	*loan_number*	*amount*
Adams	L-16	1300
Curry	L-93	500
Hayes	L-15	1500
Jackson	L-14	1500
Jones	L-17	1000
Smith	L-23	2000
Smith	L-11	900
Williams	L-17	1000

Figure 2.20 Result of $\Pi_{customer\_name,\ loan\_number,\ amount}\ (borrower \bowtie loan)$.

branch_name
Brighton
Downtown

Figure 2.22 Result of $\Pi_{branch\_name}(\sigma_{branch\_city = \text{"Brooklyn"}}(branch))$.

- Find all customers who have *both* a loan and an account at the bank.

  $$\Pi_{customer\_name}(borrower \bowtie depositor)$$

  Note that in Section 2.3.1 we wrote an expression for this query by using set intersection. We repeat this expression here.

  $$\Pi_{customer\_name}(borrower) \cap \Pi_{customer\_name}(depositor)$$

  The result relation for this query appeared earlier in Figure 2.19. This example illustrates a general fact about the relational algebra: It is possible to write several equivalent relational-algebra expressions that are quite different from one another.

- Let $r(R)$ and $s(S)$ be relations without any attributes in common; that is, $R \cap S = \emptyset$. ($\emptyset$ denotes the empty set.) Then, $r \bowtie s = r \times s$.

The *theta join* operation is an extension to the natural-join operation that allows us to combine a selection and a Cartesian product into a single operation. Consider relations $r(R)$ and $s(S)$, and let θ be a predicate on attributes in the schema $R \cup S$. The **theta join** operation $r \bowtie_\theta s$ is defined as follows:

$$r \bowtie_\theta s = \sigma_\theta(r \times s)$$

2.3.3 The Division Operation

The **division** operation, denoted by $\div$, is suited to queries that include the phrase "for all." Suppose that we wish to find all customers who have an account at *all* the branches located in Brooklyn. We can obtain all branches in Brooklyn by the expression

$$r_1 = \Pi_{branch\_name}(\sigma_{branch\_city = \text{"Brooklyn"}}(branch))$$

The result relation for this expression appears in Figure 2.22.

We can find all (*customer_name, branch_name*) pairs for which the customer has an account at a branch by writing

$$r_2 = \Pi_{customer\_name,\ branch\_name}(depositor \bowtie account)$$

Figure 2.23 shows the result relation for this expression.

Now, we need to find customers who appear in r_2 with *every* branch name in r_1. The operation that provides exactly those customers is the divide operation. We formulate the query by writing

$$\Pi_{customer\_name,\ branch\_name}(depositor \bowtie account) \div \Pi_{branch\_name}(\sigma_{branch\_city = \text{"Brooklyn"}}(branch))$$

customer_name	*branch_name*
Hayes	Perryridge
Johnson	Downtown
Johnson	Brighton
Jones	Brighton
Lindsay	Redwood
Smith	Mianus
Turner	Round Hill

Figure 2.23 Result of $\Pi_{customer\_name,\ branch\_name}\ (depositor \bowtie account)$.

The result of this expression is a relation that has the schema (*customer_name*) and that contains the tuple (Johnson).

Formally, let $r(R)$ and $s(S)$ be relations, and let $S \subseteq R$; that is, every attribute of schema S is also in schema R. The relation $r \div s$ is a relation on schema $R - S$ (that is, on the schema containing all attributes of schema R that are not in schema S). A tuple t is in $r \div s$ if and only if both of two conditions hold:

1. t is in $\Pi_{R-S}(r)$
2. For every tuple t_s in s, there is a tuple t_r in r satisfying both of the following:
   a. $t_r[S] = t_s[S]$
   b. $t_r[R - S] = t$

It may surprise you to discover that, given a division operation and the schemas of the relations, we can, in fact, define the division operation in terms of the fundamental operations. Let $r(R)$ and $s(S)$ be given, with $S \subseteq R$:

$$r \div s = \Pi_{R-S}(r) - \Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$$

To see that this expression is true, we observe that $\Pi_{R-S}(r)$ gives us all tuples t that satisfy the first condition of the definition of division. The expression on the right side of the set difference operator

$$\Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$$

serves to eliminate those tuples that fail to satisfy the second condition of the definition of division. Let us see how it does so. Consider $\Pi_{R-S}(r) \times s$. This relation is on schema R, and pairs every tuple in $\Pi_{R-S}(r)$ with every tuple in s. The expression $\Pi_{R-S,S}(r)$ merely reorders the attributes of r.

Thus, $(\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r)$ gives us those pairs of tuples from $\Pi_{R-S}(r)$ and s that do not appear in r. If a tuple t_j is in

$$\Pi_{R-S}((\Pi_{R-S}(r) \times s) - \Pi_{R-S,S}(r))$$

then there is some tuple t_s in s that does not combine with tuple t_j to form a tuple in r. Thus, t_j holds a value for attributes $R - S$ that does not appear in $r \div s$. It is these values that we eliminate from $\Pi_{R-S}(r)$.

In Section 14.4, we describe how to choose a query-evaluation plan. We can choose one based on the estimated cost of the plans. Since the cost is an estimate, the selected plan is not necessarily the least-costly plan; however, as long as the estimates are good, the plan is likely to be the least-costly one, or not much more costly than it. Such optimization, called **cost-based optimization**, is described in Section 14.4.2.

Finally, materialized views help to speed up processing of certain queries. In Section 14.5, we study how to "maintain" materialized views—that is, to keep them up to date—and how to perform query optimization with materialized views.

14.2 Transformation of Relational Expressions

A query can be expressed in several different ways, with different costs of evaluation. In this section, rather than take the relational expression as given, we consider alternative, equivalent expressions.

Two relational-algebra expressions are said to be **equivalent** if, on every legal database instance, the two expressions generate the same set of tuples. (Recall that a legal database instance is one that satisfies all the integrity constraints specified in the database schema.) Note that the order of the tuples is irrelevant; the two expressions may generate the tuples in different orders, but would be considered equivalent as long as the set of tuples is the same.

In SQL, the inputs and outputs are multisets of tuples, and a multiset version of the relational algebra is used for evaluating SQL queries. Two expressions in the *multiset* version of the relational algebra are said to be equivalent if on every legal database the two expressions generate the same multiset of tuples. The discussion in this chapter is based on the relational algebra. We leave extensions to the multiset version of the relational algebra to you as exercises.

14.2.1 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—that is 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.

We now list a number of general equivalence rules on relational-algebra expressions. Some of the equivalences listed appear in Figure 14.2. We use $\theta, \theta_1, \theta_2$, and so on to denote predicates, L_1, L_2, L_3, and so on to denote lists of attributes, and E, E_1, E_2, and so on to denote relational-algebra expressions. A relation name r is simply a special case of a relational-algebra expression, and can be used wherever E appears.

1. Conjunctive selection operations can be deconstructed into a sequence of individual selections. This transformation is referred to as a cascade of σ.

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

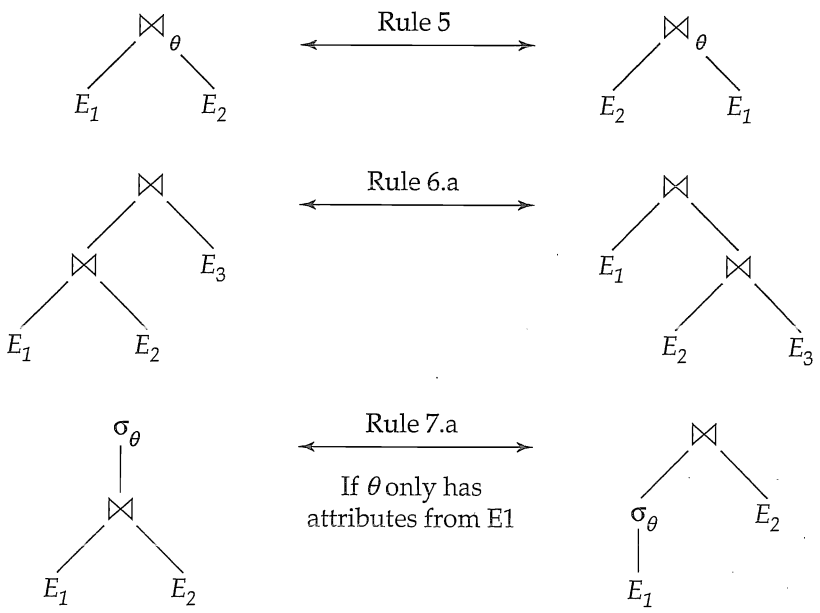

Figure 14.2 Pictorial representation of equivalences.

2. Selection operations are **commutative**.

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

3. Only the final operations in a sequence of projection operations are needed, the others can be omitted. This transformation can also be referred to as a cascade of Π.

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

4. Selections can be combined with Cartesian products and theta joins.
   a. $\sigma_\theta(E_1 \times E_2) = E_1 \bowtie_\theta E_2$
      This expression is just the definition of the theta join.
   b. $\sigma_{\theta_1}(E_1 \bowtie_{\theta_2} E_2) = E_1 \bowtie_{\theta_1 \wedge \theta_2} E_2$

5. Theta-join operations are commutative.

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

Actually, the order of attributes differs between the left-hand side and right-hand side, so the equivalence does not hold if the order of attributes is taken into account. A projection operation can be added to one of the sides of the equivalence to appropriately reorder attributes, but for simplicity we omit the projection and ignore the attribute order in most of our examples.

Recall that the natural-join operator is simply a special case of the theta-join operator; hence, natural joins are also commutative.

6. a. Natural-join operations are **associative**.

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

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 from only E_2 and E_3. Any of these conditions may be empty; hence, it follows that the Cartesian product ($\times$) operation is also associative. The commutativity and associativity of join operations are important for join reordering in query optimization.

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

a. It distributes when all the attributes in selection condition θ_0 involve only the attributes of one of the expressions (say, E_1) being joined.

$$\sigma_{\theta_0}(E_1 \bowtie_\theta E_2) = (\sigma_{\theta_0}(E_1)) \bowtie_\theta E_2$$

b. It distributes when selection condition θ_1 involves only the attributes of E_1 and θ_2 involves only the attributes of E_2.

$$\sigma_{\theta_1 \wedge \theta_2}(E_1 \bowtie_\theta E_2) = (\sigma_{\theta_1}(E_1)) \bowtie_\theta (\sigma_{\theta_2}(E_2))$$

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

a. Let L_1 and L_2 be attributes of E_1 and E_2, respectively. Suppose that the join condition θ involves only attributes in $L_1 \cup L_2$. Then,

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

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 $L_1 \cup L_2$. Then,

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

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 the union, intersection, and set-difference operations.

$$\sigma_P(E_1 - E_2) = \sigma_P(E_1) - \sigma_P(E_2)$$

Similarly, the preceding equivalence, with $-$ replaced with either $\cup$ or $\cap$, also holds. Further,

$$\sigma_P(E_1 - E_2) = \sigma_P(E_1) - E_2$$

The preceding equivalence, with $-$ replaced by $\cap$, also holds, but does not hold if $-$ is replaced by $\cup$.

12. The projection operation distributes over the union operation.

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

This is only a partial list of equivalences. More equivalences involving extended relational operators, such as the outer join and aggregation, are discussed in the exercises.

14.2.2 Examples of Transformations

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

Branch_schema = (*branch_name, branch_city, assets*)
Account_schema = (*account_number, branch_name, balance*)
Depositor_schema = (*customer_name, account_number*)

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

In our example in Section 14.1, the expression

$$\Pi_{customer\_name}(\sigma_{branch\_city = \text{"Brooklyn"}}(branch \bowtie (account \bowtie depositor)))$$

was transformed into the following expression,

$$\Pi_{customer\_name}((\sigma_{branch\_city = \text{"Brooklyn"}}(branch)) \bowtie (\overbrace{account \bowtie depositor}))$$

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.

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

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

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