3. Relational Model and Relational Algebra

Contents

- Fundamental Concepts of the Relational Model
- Integrity Constraints
- Translation ER schema \(\rightarrow\) Relational Database Schema
- Relational Algebra
- Modification of the Database

Overview

- Relational Model was introduced in 1970 by E.F. Codd (at IBM).

- Nice features: Simple and uniform data structures – \(\textit{relations}\) – and solid theoretical foundation (important for query processing and optimization)

- Relational Model is basis for most DBMSs, e.g., Oracle, Microsoft SQL Server, IBM DB2, Sybase, PostgreSQL, MySQL, . . .

- Typically used in conceptual design: either directly (creating tables using SQL DDL) or derived from a given Entity-Relationship schema.
Basic Structure of the Relational Model

- A relation $r$ over collection of sets (domain values) $D_1, D_2, \ldots, D_n$ is a subset of the Cartesian Product $D_1 \times D_2 \times \ldots \times D_n$
  A relation thus is a set of $n$-tuples $(d_1, d_2, \ldots, d_n)$ where $d_i \in D_i$.

- Given the sets
  
  StudId = \{412, 307, 540\}
  StudName = \{Smith, Jones\}
  Major = \{CS, CSE, BIO\}

  then $r = \{(412, Smith, CS), (307, Jones, CSE), (412, Smith, CSE)\}$ is a relation over StudId $\times$ StudName $\times$ Major

Relation Schema, Database Schema, and Instances

- Let $A_1, A_2, \ldots, A_n$ be attribute names with associated domains $D_1, D_2, \ldots, D_n$, then
  
  $R(A_1: D_1, A_2: D_2, \ldots, A_n: D_n)$

  is a relation schema. For example,
  
  Student(StudId: integer, StudName: string, Major: string)

- A relation schema specifies the name and the structure of the relation.

- A collection of relation schemas is called a relational database schema.
Relation Schema, Database Schema, and Instances

- A relation instance \( r(R) \) of a relation schema can be thought of as a table with \( n \) columns and a number of rows. Instead of relation instance we often just say relation. An instance of a database schema thus is a collection of relations.

- An element \( t \in r(R) \) is called a tuple (or row).

<table>
<thead>
<tr>
<th>Student</th>
<th>StudId</th>
<th>StudName</th>
<th>Major</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>412</td>
<td>Smith</td>
<td>CS</td>
</tr>
<tr>
<td></td>
<td>307</td>
<td>Jones</td>
<td>CSE</td>
</tr>
<tr>
<td></td>
<td>412</td>
<td>Smith</td>
<td>CSE</td>
</tr>
</tbody>
</table>

- A relation has the following properties:
  - the order of rows is irrelevant, and
  - there are no duplicate rows in a relation

Integrity Constraints in the Relational Model

- Integrity constraints (ICs): must be true for any instance of a relation schema (admissible instances)
  - ICs are specified when the schema is defined
  - ICs are checked by the DBMS when relations (instances) are modified
- If DBMS checks ICs, then the data managed by the DBMS more closely correspond to the real-world scenario that is being modeled!
Primary Key Constraints

- A set of attributes is a *key* for a relation if:
  1. no two distinct tuples have the same values for all key attributes, and
  2. this is not true for any subset of that key.

- If there is more than one key for a relation (i.e., we have a set of candidate keys), one is chosen (by the designer or DBA) to be the *primary key*.
  
  `Student(StudId : number, StudName : string, Major : string)`

- For candidate keys not chosen as primary key, *uniqueness* constraints can be specified.

- Note that it is often useful to introduce an artificial primary key (as a single attribute) for a relation, in particular if this relation is often “ referenced”.

3. Relational Model and Relational Algebra
Foreign Key Constraints and Referential Integrity

- Set of attributes in one relation (child relation) that is used to “refer” to a tuple in another relation (parent relation). Foreign key must refer to the primary key of the referenced relation.

- Foreign key attributes are required in relation schemas that have been derived from relationship types. Example:

  offers(Prodname → PRODUCTS, SName → SUPPLIERS, Price)
  orders((FName, LName) → CUSTOMERS, SName → SUPPLIERS, Prodname → PRODUCTS, Quantity)

  Foreign/primary key attributes must have matching domains.

- A foreign key constraint is **satisfied** for a tuple if either
  - some values of the foreign key attributes are *null* (meaning a reference is not known), or
  - the values of the foreign key attributes occur as the values of the primary key (of some tuple) in the parent relation.

- The combination of foreign key attributes in a relation schema typically builds the primary key of the relation, e.g.,

  offers(Prodname → PRODUCTS, SName → SUPPLIERS, Price)

- If all foreign key constraints are enforced for a relation, *referential integrity* is achieved, i.e., there are no dangling references.
Translation of an ER Schema into a Relational Schema

1. Entity type \( E(A_1, \ldots, A_n, B_1, \ldots, B_m) \)
   \[ \implies \text{relation schema } E(A_1, \ldots, A_n, B_1, \ldots, B_m). \]

2. Relationship type \( R(E_1, \ldots, E_n, A_1, \ldots, A_m) \)
   with participating entity types \( E_1, \ldots, E_n; \)
   \( X_i \equiv \text{foreign key attribute(s) referencing primary key attribute(s) of} \)
   \( \text{relation schema corresponding to } E_i. \)
   \[ \implies R(X_1 \rightarrow E_1, \ldots, X_n \rightarrow E_n, A_1, \ldots, A_m) \]

For a functional relationship (N:1, 1:N), an optimization is possible. Assume N:1 relationship type between \( E_1 \) and \( E_2. \)
We can extend the schema of \( E_1 \) to
\[ E_1(A_1, \ldots, A_n, X_2 \rightarrow E_2, B_1, \ldots, B_m), \text{ e.g., } \]

\[ \text{EMPLOYEES(EmpId, DeptNo \rightarrow DEPARTMENTS, \ldots)} \]
• Example translation:

![Diagram of the entities and relationships]

• According to step 1:

  BOOKS(DocId, Title, Publisher, Year)
  STUDENTS(StId, StName, Major, Year)
  DESCRIPTIONS(Keyword)
  AUTHORS(AName, Address)

In step 2 the relationship types are translated:

  borrows(DocId → BOOKS, StId → STUDENTS, Date)
  has-written(DocId → BOOKS, AName → AUTHORS)
  describes(DocId → BOOKS, Keyword → DESCRIPTIONS)

No need for extra relation for entity type “DESCRIPTIONS”:

  Descriptions(DocId → BOOKS, Keyword)
3.2 Relational Algebra

Query Languages

- A query language (QL) is a language that allows users to manipulate and retrieve data from a database.

- The relational model supports simple, powerful QLs (having strong formal foundation based on logics, allow for much optimization)

- Query Language ≠ Programming Language
  - QLs are not expected to be Turing-complete, not intended to be used for complex applications/computations
  - QLs support easy access to large data sets

- Categories of QLs: procedural versus declarative

- Two (mathematical) query languages form the basis for “real” languages (e.g., SQL) and for implementation
  - *Relational Algebra*: procedural, very useful for representing query execution plans, and query optimization techniques.
  - *Relational Calculus*: declarative, logic based language

- Understanding algebra (and calculus) is the key to understanding SQL, query processing and optimization.
Relational Algebra

- Procedural language

- Queries in relational algebra are applied to relation instances, result of a query is again a relation instance

- Six basic operators in relational algebra:
  - \( \text{select} \) \( \sigma \) selects a subset of tuples from reln
  - \( \text{project} \) \( \pi \) deletes unwanted columns from reln
  - \( \text{Cartesian Product} \) \( \times \) allows to combine two relations
  - \( \text{Set-difference} \) \( - \) tuples in reln. 1, but not in reln. 2
  - \( \text{Union} \) \( \cup \) tuples in reln 1 plus tuples in reln 2
  - \( \text{Rename} \) \( \rho \) renames attribute(s) and relation

- The operators take one or two relations as input and give a new relation as a result (relational algebra is “closed”).
**Select Operation**

- Notation: $\sigma_P(r)$

  Defined as
  
  $$\sigma_P(r) := \{ t \mid t \in r \text{ and } P(t) \}$$

  where
  - $r$ is a relation (name),
  - $P$ is a formula in propositional calculus, composed of conditions of the form
    
    $$<\text{attribute}> = <\text{attribute}> \text{ or } <\text{constant}>$$

    Instead of “=” any other comparison predicate is allowed ($\neq, <, >$ etc).

    Conditions can be composed through $\land$ (and), $\lor$ (or), $\neg$ (not)

- Example: given the relation $r$

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>$\alpha$</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>$\beta$</td>
<td>5</td>
<td>7</td>
</tr>
<tr>
<td>$\beta$</td>
<td>$\beta$</td>
<td>12</td>
<td>3</td>
</tr>
<tr>
<td>$\beta$</td>
<td>$\beta$</td>
<td>23</td>
<td>10</td>
</tr>
</tbody>
</table>

  $\sigma_{A=B \land D>5}(r)$

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>$\alpha$</td>
<td>1</td>
<td>7</td>
</tr>
<tr>
<td>$\beta$</td>
<td>$\beta$</td>
<td>23</td>
<td>10</td>
</tr>
</tbody>
</table>
**Project Operation**

- Notation: $\pi_{A_1, A_2, \ldots, A_k}(r)$
  
  where $A_1, \ldots, A_k$ are attribute names and $r$ is a relation (name).

- The result of the projection operation is defined as the relation that has $k$ columns obtained by erasing all columns from $r$ that are not listed.

- Duplicate rows are removed from result because relations are sets.

- Example: given the relations $r$

  $r$

  $\begin{array}{ccc}
  A & B & C \\
  \alpha & 10 & 2 \\
  \alpha & 20 & 2 \\
  \beta & 30 & 2 \\
  \beta & 40 & 4 \\
  \end{array}$

  $\pi_{A, C}(r)$

  $\begin{array}{cc}
  A & C \\
  \alpha & 2 \\
  \beta & 2 \\
  \beta & 4 \\
  \end{array}$
**Cartesian Product**

- Notation: $r \times s$ where both $r$ and $s$ are relations
  Defined as $r \times s := \{ tq \mid t \in r \text{ and } q \in s \}$

- Assume that attributes of $r(R)$ and $s(S)$ are disjoint, i.e., $R \cap S = \emptyset$.
  If attributes of $r(R)$ and $s(S)$ are not disjoint, then the rename operation must be applied first.

- Example: relations $r$, $s$:

  $$
  \begin{array}{c|c}
  r & A & B \\
  \hline
  \alpha & 1 \\
  \beta & 2 \\
  \end{array}
  \quad
  \begin{array}{c|c|c}
  s & C & D & E \\
  \hline
  \alpha & 10 & + \\
  \beta & 10 & + \\
  \beta & 20 & - \\
  \gamma & 10 & - \\
  \end{array}
  \quad
  \begin{array}{c|c|c|c|c}
  r \times s & A & B & C & D & E \\
  \hline
  \alpha & 1 & \alpha & 10 & + \\
  \alpha & 1 & \beta & 10 & + \\
  \alpha & 1 & \beta & 20 & - \\
  \alpha & 1 & \gamma & 10 & - \\
  \beta & 2 & \alpha & 10 & + \\
  \beta & 2 & \beta & 10 & + \\
  \beta & 2 & \beta & 20 & - \\
  \beta & 2 & \gamma & 10 & - \\
  \end{array}
  $$
**Union Operator**

- Notation: \( r \cup s \) where both \( r \) and \( s \) are relations

  Defined as \( r \cup s := \{ t \mid t \in r \text{ or } t \in s \} \)

- For \( r \cup s \) to be applicable,
  1. \( r, s \) must have the same number of attributes
  2. Attribute domains must be compatible (e.g., 3rd column of \( r \) has a data type matching the data type of the 3rd column of \( s \))

- Example: given the relations \( r \) and \( s \)

\[
\begin{array}{ccc}
\text{r} & \text{A} & \text{B} \\
\alpha & 1 \\
\alpha & 2 \\
\beta & 1 \\
\end{array}
\quad
\begin{array}{ccc}
\text{s} & \text{A} & \text{B} \\
\alpha & 2 \\
\beta & 3 \\
\end{array}
\]

\[
\begin{array}{ccc}
\text{r} \cup \text{s} & \text{A} & \text{B} \\
\alpha & 1 \\
\alpha & 2 \\
\beta & 1 \\
\beta & 3 \\
\end{array}
\]

3. Relational Model and Relational Algebra
Set Difference Operator

- Notation: $r - s$ where both $r$ and $s$ are relations
  Defined as $r - s := \{ t \mid t \in r \text{ and } t \notin s \}$

- For $r - s$ to be applicable,
  1. $r$ and $s$ must have the same arity
  2. Attribute domains must be compatible

- Example: given the relations $r$ and $s$

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r$</td>
<td>$\alpha$</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>$\alpha$</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>$\beta$</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s$</td>
<td>$\alpha$</td>
<td>2</td>
</tr>
<tr>
<td></td>
<td>$\beta$</td>
<td>3</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>A</th>
<th>B</th>
</tr>
</thead>
<tbody>
<tr>
<td>$r - s$</td>
<td>$\alpha$</td>
<td>1</td>
</tr>
<tr>
<td></td>
<td>$\beta$</td>
<td>1</td>
</tr>
</tbody>
</table>
**Rename Operation**

- Allows to name and therefore to refer to the result of relational algebra expression.

- Allows to refer to a relation by more than one name (e.g., if the same relation is used twice in a relational algebra expression).

- Example:

\[ \rho_x(E) \]

returns the relational algebra expression \( E \) under the name \( x \)

If a relational algebra expression \( E \) (which is a relation) has the arity \( k \), then

\[ \rho_x(A_1, A_2, \ldots, A_k)(E) \]

returns the expression \( E \) under the name \( x \), and with the attribute names \( A_1, A_2, \ldots, A_k \).
Composition of Operations

- It is possible to build relational algebra expressions using multiple operators similar to the use of arithmetic operators (nesting of operators)

- Example: $\sigma_{A=C}(r \times s)$

\[
\begin{array}{|c|c|c|c|c|}
\hline
A & B & C & D & E \\
\hline
\alpha & 1 & \alpha & 10 & + \\
\alpha & 1 & \beta & 10 & + \\
\alpha & 1 & \beta & 20 & - \\
\alpha & 1 & \gamma & 10 & - \\
\beta & 2 & \alpha & 10 & + \\
\beta & 2 & \beta & 10 & + \\
\beta & 2 & \beta & 20 & - \\
\beta & 2 & \gamma & 10 & - \\
\hline
\end{array}
\]

\[
\begin{array}{|c|c|c|c|c|}
\hline
A & B & C & D & E \\
\hline
\alpha & 1 & \alpha & 10 & + \\
\beta & 2 & \beta & 10 & + \\
\beta & 2 & \beta & 20 & - \\
\hline
\end{array}
\]
Example Queries

Assume the following relations:

- **BOOKS** (DocId, Title, Publisher, Year)
- **STUDENTS** (StId, StName, Major, Age)
- **AUTHORS** (AName, Address)
- **borrows** (DocId, StId, Date)
- **has-written** (DocId, AName)
- **describes** (DocId, Keyword)

- List the year and title of each book.
  \[ \pi_{\text{Year}, \text{Title}}(\text{BOOKS}) \]

- List all information about students whose major is CS.
  \[ \sigma_{\text{Major} = 'CS'}(\text{STUDENTS}) \]

- List all students with the books they can borrow.
  \[ \text{STUDENTS} \times \text{BOOKS} \]

  \[ \sigma_{\text{Publisher} = 'McGraw-Hill' \land \text{Year} < 1990}(\text{BOOKS}) \]
• *List the name of those authors who are living in Davis.*

\[ \pi_{\text{AName}}(\sigma_{\text{Address like 'Davis'}}(\text{AUTHORS})) \]

• *List the name of students who are older than 30 and who are not studying CS.*

\[ \pi_{\text{StName}}(\sigma_{\text{Age}>30}(\text{STUDENTS})) - \pi_{\text{StName}}(\sigma_{\text{Major}='CS'}(\text{STUDENTS})) \]

• *Rename AName in the relation AUTHORS to Name.*

\[ \rho_{\text{AUTHORS}}(\text{Name, Address})(\text{AUTHORS}) \]
Composed Queries (formal definition)

- A **basic expression** in the relational algebra consists of either of the following:
  - A relation in the database
  - A constant relation
    (fixed set of tuples, e.g., \{(1, 2), (1, 3), (2, 3)\})

- If $E_1$ and $E_2$ are expressions of the relational algebra, then the following expressions are relational algebra expressions, too:
  - $E_1 \cup E_2$
  - $E_1 \setminus E_2$
  - $E_1 \times E_2$
  - $\sigma_P(E_1)$ where $P$ is a predicate on attributes in $E_1$
  - $\pi_A(E_1)$ where $A$ is a list of some of the attributes in $E_1$
  - $\rho_x(E_1)$ where $x$ is the new name for the result relation [and its attributes] determined by $E_1$
Examples of Composed Queries

1. List the names of all students who have borrowed a book and who are CS majors.

   \[ \pi_{\text{StName}} \left( \sigma_{\text{STUDENTS.StId}=\text{borrows.StId}} \left( \sigma_{\text{Major}=\text{'CS'}} \left( \text{STUDENTS} \times \text{borrows} \right) \right) \right) \]

2. List the title of books written by the author 'Silberschatz'.

   \[ \pi_{\text{Title}} \left( \sigma_{\text{AName}=\text{'Silberschatz'}} \left( \sigma_{\text{has-written.DocId}=\text{BOOKS.DocID}} \left( \text{has-written} \times \text{BOOKS} \right) \right) \right) \]

   or

   \[ \pi_{\text{Title}} \left( \sigma_{\text{has-written.DocId}=\text{BOOKS.DocID}} \left( \sigma_{\text{AName}=\text{'Silberschatz'}} \left( \text{has-written} \times \text{BOOKS} \right) \right) \right) \]

3. As 2., but not books that have the keyword 'database'.

   ... as for 2. ... 

   \[ \pi_{\text{Title}} \left( \sigma_{\text{describes.DocId}=\text{BOOKS.DocID}} \left( \sigma_{\text{Keyword}=\text{'database'}} \left( \text{describes} \times \text{BOOKS} \right) \right) \right) \]

4. Find the name of the youngest student.

   \[ \pi_{\text{StName}} \left( \text{STUDENTS} \right) - 
   \pi_{\text{S1.StName}} \left( \sigma_{\text{S1.Age}>\text{S2.Age}} \left( \rho_{\text{S1}} \left( \text{STUDENTS} \times \rho_{\text{S2}} \left( \text{STUDENTS} \right) \right) \right) \right) \]

5. Find the title of the oldest book.

   \[ \pi_{\text{Title}} \left( \text{BOOKS} \right) - 
   \pi_{\text{B1.Title}} \left( \sigma_{\text{B1.Year}>\text{B2.Year}} \left( \rho_{\text{B1}} \left( \text{BOOKS} \times \rho_{\text{B2}} \left( \text{BOOKS} \right) \right) \right) \right) \]
Additional Operators

These operators do not add any power (expressiveness) to the relational algebra but simplify common (often complex and lengthy) queries.

- **Set-Intersection** \( \cap \)
- **Natural Join** \( \times \)
- **Condition Join** \( \times_C \) (also called Theta-Join)
- **Division** \( \div \)
- **Assignment** \( \leftarrow \)

**Set-Intersection**

- Notation: \( r \cap s \)
  Defined as \( r \cap s := \{ t \mid t \in r \text{ and } t \in s \} \)

- For \( r \cap s \) to be applicable,
  1. \( r \) and \( s \) must have the same arity
  2. Attribute domains must be compatible

- Derivation: \( r \cap s = r - (r - s) \)

- Example: given the relations \( r \) and \( s \)

<table>
<thead>
<tr>
<th>( r )</th>
<th>( s )</th>
<th>( r \cap s )</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \alpha )</td>
<td>1</td>
<td>( \alpha )</td>
</tr>
<tr>
<td>( \alpha )</td>
<td>2</td>
<td>( \beta )</td>
</tr>
<tr>
<td>( \beta )</td>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>
**Natural Join**

- Notation: \( r \Join s \)

- Let \( r, s \) be relations on schemas \( R \) and \( S \), respectively. The result is a relation on schema \( R \cup S \). The result tuples are obtained by considering each pair of tuples \( t_r \in r \) and \( t_s \in s \).

- If \( t_r \) and \( t_s \) have the same value for each of the attributes in \( R \cap S \) ("same name attributes"), a tuple \( t \) is added to the result such that
  - \( t \) has the same value as \( t_r \) on \( r \)
  - \( t \) has the same value as \( t_s \) on \( s \)

- Example: Given the relations \( R(A, B, C, D) \) and \( S(B, D, E) \)
  - Join can be applied because \( R \cap S \neq \emptyset \)
  - the result schema is \( (A, B, C, D, E) \)
  - and the result of \( r \Join s \) is defined as
    \[
    \pi_{r.A, r.B, r.C, r.D, s.E}(\sigma_{r.B = s.B \land r.D = s.D}(r \times s))
    \]
• Example: given the relations $r$ and $s$

\[
\begin{array}{cccc}
\alpha & 1 & \alpha & a \\
\beta  & 2 & \gamma & a \\
\gamma & 4 & \beta  & b \\
\alpha & 1 & \gamma & a \\
\delta & 2 & \beta  & b \\
\end{array}
\quad
\begin{array}{ccc}
1 & a & \alpha \\
3 & a & \beta \\
1 & a & \gamma \\
2 & b & \delta \\
3 & b & \tau \\
\end{array}
\]

\[
\begin{array}{cccc}
\alpha & 1 & \alpha & a & \alpha \\
\alpha & 1 & \alpha & a & \gamma \\
\alpha & 1 & \gamma & a & \alpha \\
\alpha & 1 & \gamma & a & \gamma \\
\delta & 2 & \beta  & b & \delta \\
\end{array}
\]

3. Relational Model and Relational Algebra
**Condition Join**

- Notation: \( r \bowtie_C s \)

\( C \) is a condition on attributes in \( R \cup S \), result schema is the same as that of Cartesian Product. If \( R \cap S \neq \emptyset \) and condition \( C \) refers to these attributes, some of these attributes must be renamed.

Sometimes also called \textit{Theta Join} \((r \bowtie_{\theta} s)\).

- Derivation: \( r \bowtie_C s = \sigma_C(r \times s) \)

- Note that \( C \) is a condition on attributes from both \( r \) and \( s \)

- Example: given two relations \( r, s \)

\[
\begin{array}{cccc}
A & B & C \\
1 & 2 & 3 \\
4 & 5 & 6 \\
7 & 8 & 9 \\
\end{array}
\quad
\begin{array}{cccc}
D & E \\
3 & 1 \\
6 & 2 \\
\end{array}
\]

\[
\begin{array}{cccccc}
A & B & C & D & E \\
1 & 2 & 3 & 3 & 1 \\
1 & 2 & 3 & 6 & 2 \\
4 & 5 & 6 & 6 & 2 \\
\end{array}
\]
If \( C \) involves only the comparison operator “=”, the condition join is also called \textit{Equi-Join}.

- Example 2:

\[
\begin{array}{|c|c|c|}
\hline
A & B & C \\
\hline
4 & 5 & 6 \\
7 & 8 & 9 \\
\hline
\end{array}
\quad
\begin{array}{|c|c|}
\hline
C & D \\
\hline
6 & 8 \\
10 & 12 \\
\hline
\end{array}
\]

\[
r \Join_{C=SC} (\rho_{S(SC,D)}(s))
\]

\[
\begin{array}{|c|c|c|c|c|}
\hline
A & B & C & SC & D \\
\hline
4 & 5 & 6 & 6 & 8 \\
\hline
\end{array}
\]
**Division**

- **Notation:** \( r \div s \)

- **Precondition:** attributes in \( S \) must be a subset of attributes in \( R \), i.e., \( S \subseteq R \). Let \( r, s \) be relations on schemas \( R \) and \( S \), respectively, where

\[
\begin{align*}
- & R(A_1, \ldots, A_m, B_1, \ldots, B_n) \\
- & S(B_1, \ldots, B_n)
\end{align*}
\]

The result of \( r \div s \) is a relation on schema \( R - S = (A_1, \ldots, A_m) \)

- **Suited for queries that include the phrase “for all”.**

The result of the division operator consists of the set of tuples from \( r \) defined over the attributes \( R - S \) that match the combination of every tuple in \( s \).

\[
r \div s := \{ t \mid t \in \pi_{R-S}(r) \land \forall u \in s: tu \in r \}\]

3. Relational Model and Relational Algebra
• Example: given the relations $r, s$:

$$r$$

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>a</td>
<td>$\alpha$</td>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>a</td>
<td>$\gamma$</td>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>$\alpha$</td>
<td>a</td>
<td>$\gamma$</td>
<td>b</td>
<td>1</td>
</tr>
<tr>
<td>$\beta$</td>
<td>a</td>
<td>$\gamma$</td>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>$\beta$</td>
<td>a</td>
<td>$\gamma$</td>
<td>b</td>
<td>3</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>a</td>
<td>$\gamma$</td>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>a</td>
<td>$\gamma$</td>
<td>b</td>
<td>1</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>a</td>
<td>$\beta$</td>
<td>b</td>
<td>1</td>
</tr>
</tbody>
</table>

$$s$$

<table>
<thead>
<tr>
<th>D</th>
<th>E</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>1</td>
</tr>
<tr>
<td>b</td>
<td>1</td>
</tr>
</tbody>
</table>

$$r \div s$$

<table>
<thead>
<tr>
<th>A</th>
<th>B</th>
<th>C</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\alpha$</td>
<td>a</td>
<td>$\gamma$</td>
</tr>
<tr>
<td>$\gamma$</td>
<td>a</td>
<td>$\gamma$</td>
</tr>
</tbody>
</table>
Assignment

- Operation ($\leftarrow$) that provides a convenient way to express complex queries.
  Idea: write query as sequential program consisting of a series of assignments followed by an expression whose value is “displayed” as the result of the query.

- Assignment must always be made to a temporary relation variable.
  The result to the right of $\leftarrow$ is assigned to the relation variable on the left of the $\leftarrow$. This variable may be used in subsequent expressions.

Example Queries

1. *List each book with its keywords.*
   
   BOOKS $\Join$ Descriptions

   Note that books having no keyword are not in the result.

2. *List each student with the books s/he has borrowed.*
   
   BOOKS $\Join$ (borrows $\Join$ STUDENTS)
3. List the title of books written by the author 'Ullman'.

\[ \pi_{\text{Title}}(\sigma_{\text{AName} = 'Ullman'}(\text{BOOKS} \bowtie \text{has-written})) \]

or

\[ \pi_{\text{Title}}(\text{BOOKS} \bowtie \sigma_{\text{AName} = 'Ullman'}(\text{has-written})) \]

4. List the authors of the books the student 'Smith' has borrowed.

\[ \pi_{\text{AName}}(\sigma_{\text{StName} = 'Smith'}(\text{has-written} \bowtie (\text{borrows} \bowtie \text{STUDENTS}))) \]

5. Which books have both keywords 'database' and 'programming'?

\[ \text{BOOKS} \bowtie (\pi_{\text{DocId}}(\sigma_{\text{Keyword} = 'database'}(\text{Descriptions}))) \cap \pi_{\text{DocId}}(\sigma_{\text{Keyword} = 'programming'}(\text{Descriptions}))) \]

or

\[ \sigma_{\text{Keyword} \in \{('database'), ('programming')\}}(\text{BOOKS} \bowtie (\text{Descriptions} \div \{('database'), ('programming')\})) \]

with \{('database'), ('programming')\} being a constant relation.

6. Query 4 using assignments.

\[ \text{temp1} \leftarrow \text{borrows} \bowtie \text{STUDENTS} \]
\[ \text{temp2} \leftarrow \text{has-written} \bowtie \text{temp1} \]
\[ \text{result} \leftarrow \pi_{\text{AName}}(\sigma_{\text{StName} = 'Smith'}(\text{temp2})) \]
Modifications of the Database

- The content of the database may be modified using the operations `insert`, `delete` or `update`.
- Operations can be expressed using the assignment operator.
  \[ r_{new} \leftarrow \text{operations on}(r_{old}) \]

**Insert**

- Either specify tuple(s) to be inserted, or write a query whose result is a set of tuples to be inserted.
- \[ r \leftarrow r \cup E, \text{ where } r \text{ is a relation and } E \text{ is a relational algebra expression.} \]
- \[ \text{STUDENTS} \leftarrow \text{STUDENTS} \cup \{(1024, 'Clark', 'CSE', 26)\} \]

**Delete**

- Analogous to insert, but \(-\) operator instead of \(\cup\) operator.
- Can only delete whole tuples, cannot delete values of particular attributes.
- \[ \text{STUDENTS} \leftarrow \text{STUDENTS} - (\sigma_{\text{major}='CS'}(\text{STUDENTS})) \]

**Update**

- Can be expressed as sequence of delete and insert operations. Delete operation deletes tuples with their old value(s) and insert operation inserts tuples with their new value(s).