3. Relational Model and Relational Algebra

Contents

- Fundamental Concepts of the Relational Model
- Integrity Constraints
- Translation ER schema $\longrightarrow$ 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 – *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 × StudName × 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, ..., D_n, then

  $$R(A_1\colon D_1, A_2\colon D_2, \ldots, A_n\colon D_n)$$

  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).

Student	StudId	StudName	Major	
				$\longleftarrow$ relation schema
	412	Smith	CS	
	307	Jones	CSE	$\longleftarrow$ tuple
	412	Smith	CSE	

- 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($\underline{\text{StudId}}$: 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".

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(<u>Prodname → PRODUCTS, SName → SUPPLIERS</u>, 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 $E(\underline{A_1, \ldots, A_n}, B_1, \ldots, B_m)$

   $\Longrightarrow$ relation schema $E(\underline{A_1, \ldots, A_n}, B_1, \ldots, B_m)$.

2. Relationship type $R(E_1, \ldots, E_n, A_1, \ldots, A_m)$
   with participating entity types $E_1, \ldots, E_n$;
   $X_i \equiv$ foreign key attribute(s) referencing primary key attribute(s) of
   relation schema corresponding to E_i.

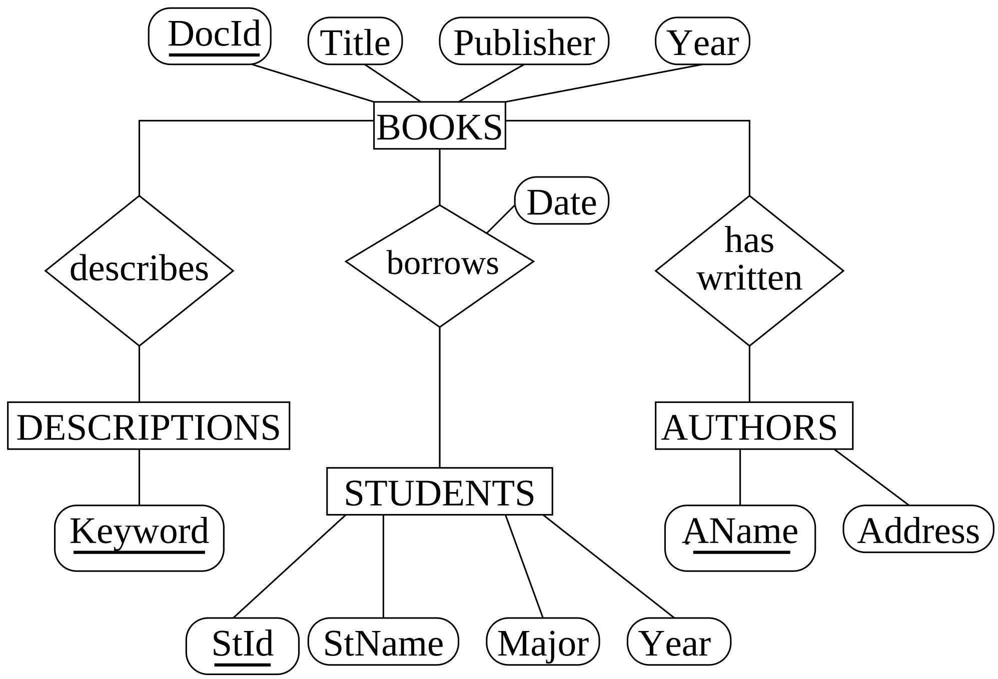
   $\Longrightarrow$ $R(\underline{X_1} \rightarrow E_1, \ldots, \underline{X_n} \rightarrow E_n, A_1, \ldots, A_m)$

   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

   $$E_1(\underline{A_1, \ldots, A_n}, X_2 \rightarrow E_2, B_1, \ldots, B_m), \text{ e.g.,}$$

   EMPLOYEES(<u>EmpId</u>, DeptNo → DEPARTMENTS, . . .)

- Example translation:



- According to step 1:

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

  In step 2 the relationship types are translated:

  borrows(<u>DocId</u> $\longrightarrow$ BOOKS, <u>StId</u> $\longrightarrow$ STUDENTS, Date)
  has-written(<u>DocId</u> $\longrightarrow$ BOOKS, <u>AName</u> $\longrightarrow$ AUTHORS)
  describes(<u>DocId</u> $\longrightarrow$ BOOKS, <u>Keyword</u> $\longrightarrow$ DESCRIPTIONS)

  No need for extra relation for entity type "DESCRIPTIONS":

  Descriptions(<u>DocId</u> $\longrightarrow$ BOOKS, <u>Keyword</u>)

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:

select	σ	selects a subset of tuples from reln
project	π	deletes unwanted columns from reln
Cartesian Product	$\times$	allows to combine two relations
Set-difference	$-$	tuples in reln. 1, but not in reln. 2
Union	$\cup$	tuples in reln 1 plus tuples in reln 2
Rename	ρ	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 $\wedge$ (**and**), $\vee$ (**or**), $\neg$ (**not**)

- Example: given the relation r

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

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

A	B	C	D
α	α	1	7
β	β	23	10

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

A	B	C
α	10	2
α	20	2
β	30	2
β	40	4

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

A	C
α	2
β	2
β	4

Cartesian Product

- Notation: $r \times s$ where both r and s are relations

  Defined as $r \times s := \{tq \mid t \in r \textbf{ and } q \in s\}$

- 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:

r

A	B
α	1
β	2

s

C	D	E
α	10	$+$
β	10	$+$
β	20	$-$
γ	10	$-$

$r \times s$

A	B	C	D	E
α	1	α	10	$+$
α	1	β	10	$+$
α	1	β	20	$-$
α	1	γ	10	$-$
β	2	α	10	$+$
β	2	β	10	$+$
β	2	β	20	$-$
β	2	γ	10	$-$

Union Operator

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

  Defined as $r \cup s := \{t \mid t \in r \textbf{ or } t \in s\}$

- 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

r

A	B
α	1
α	2
β	1

s

A	B
α	2
β	3

$r \cup s$

A	B
α	1
α	2
β	1
β	3

Set Difference Operator

- Notation: $r - s$ where both r and s are relations

  Defined as $r - s := \{t \mid t \in r \textbf{ and } t \notin s\}$

- 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

r

A	B
α	1
α	2
β	1

s

A	B
α	2
β	3

$r - s$

A	B
α	1
β	1

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

$r \times s$

A	B	C	D	E
α	1	α	10	$+$
α	1	β	10	$+$
α	1	β	20	$-$
α	1	γ	10	$-$
β	2	α	10	$+$
β	2	β	10	$+$
β	2	β	20	$-$
β	2	γ	10	$-$

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

A	B	C	D	E
α	1	α	10	$+$
β	2	β	10	$+$
β	2	β	20	$-$

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.*

  $\pi_{\text{Year, Title}}(\text{BOOKS})$

- *List all information about students whose major is CS.*

  $\sigma_{\text{Major} = \text{'CS'}}(\text{STUDENTS})$

- *List all students with the books they can borrow.*

  $\text{STUDENTS} \times \text{BOOKS}$

- *List all books published by McGraw-Hill before 1990.*

  $\sigma_{\text{Publisher} = \text{'McGraw-Hill'} \wedge \text{Year} < 1990}(\text{BOOKS})$

- *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`.

  $\rho_{\text{AUTHORS(Name, Address)}}(\text{AUTHORS})$

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:
  - $E_1 \cup E_2$
  - $E_1 - E_2$
  - $E_1 \times E_2$
  - $\sigma_P(E_1)$ 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.*

   $$\pi_{\text{StName}}(\sigma_{\text{STUDENTS.StId=borrows.StId}}$$
   $$(\sigma_{\text{Major='CS'}}(\text{STUDENTS}) \times \text{borrows}))$$

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

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

   or

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

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

   . . . as for 2. . . .

   $$- \ \pi_{\text{Title}}(\sigma_{\text{describes.DocId=BOOKS.DocId}}$$
   $$(\sigma_{\text{Keyword='database'}}(\text{describes}) \times \text{BOOKS}))$$

4. *Find the name of the youngest student.*

   $$\pi_{\text{StName}}(\text{STUDENTS}) -$$
   $$\pi_{\text{S1.StName}}(\sigma_{\text{S1.Age>S2.Age}}(\rho_{\text{S1}}(\text{STUDENTS}) \times \rho_{\text{S2}}(\text{STUDENTS})))$$

5. *Find the title of the oldest book.*

   $$\pi_{\text{Title}}(\text{BOOKS}) -$$
   $$\pi_{\text{B1.Title}}(\sigma_{\text{B1.Year>B2.Year}}(\rho_{\text{B1}}(\text{BOOKS}) \times \rho_{\text{B2}}(\text{BOOKS})))$$

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	$\bowtie$	
Condition Join	$\bowtie_C$	(also called Theta-Join)
Division	$\div$	
Assignment	$\longleftarrow$	

Set-Intersection

- Notation: $r \cap s$
  Defined as $r \cap s := \{t \mid t \in r \textbf{ and } t \in s\}$
- 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

r

A	B
α	1
α	2
β	1

s

A	B
α	2
β	3

$r \cap s$

A	B
α	2

Natural Join

- Notation: $r \bowtie s$
- 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 $r \bowtie s$ is defined as

$$\pi_{r.A,r.B,r.C,r.D,s.E}(\sigma_{r.B=s.B \wedge r.D=s.D}(r \times s))$$

- Example: given the relations r and s

r

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

s

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	τ

$r \bowtie s$

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

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 *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

r

A	B	C
1	2	3
4	5	6
7	8	9

s

D	E
3	1
6	2

$r \bowtie_{\mathrm{B<D}} s$

A	B	C	D	E
1	2	3	3	1
1	2	3	6	2
4	5	6	6	2

If C involves only the comparison operator "$=$", the condition join is also called *Equi-Join.*

- Example 2:

r

A	B	C
4	5	6
7	8	9

s

C	D
6	8
10	12

$r \bowtie_{\mathsf{C=SC}} (\rho_{\mathsf{S(SC,D)}}(s))$

A	B	C	SC	D
4	5	6	6	8

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
  - $R(A_1, \ldots, A_m, B_1, \ldots, B_n)$
  - $S(B_1, \ldots, B_n)$

  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.

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

- Example: given the relations r, s:

r

A	B	C	D	E
α	a	α	a	1
α	a	γ	a	1
α	a	γ	b	1
β	a	γ	a	1
β	a	γ	b	3
γ	a	γ	a	1
γ	a	γ	b	1
γ	a	β	b	1

s

D	E
a	1
b	1

$r \div s$

A	B	C
α	a	γ
γ	a	γ

Assignment

- Operation ($\longleftarrow$) 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 $\longleftarrow$ is assigned to the relation variable on the left of the $\longleftarrow$. This variable may be used in subsequent expressions.

Example Queries

1. *List each book with its keywords.*

   BOOKS $\bowtie$ Descriptions

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

2. *List each student with the books s/he has borrowed.*

   BOOKS $\bowtie$ (borrows $\bowtie$ 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

   $\text{BOOKS} \bowtie (\text{Descriptions} \div \{(\text{'database'}), (\text{'programming'})\})$

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

6. Query 4 using assignments.

   temp1 $\longleftarrow$ borrows $\bowtie$ STUDENTS

   temp2 $\longleftarrow$ has-written $\bowtie$ temp1

   result $\longleftarrow$ $\pi_{\text{AName}}(\sigma_{\text{StName='Smith'}}(\text{temp2}))$

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. $r_{new} \longleftarrow$ 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.
- $r \longleftarrow r \cup E$, where r is a relation and E is a relational algebra expression.
- STUDENTS $\longleftarrow$ STUDENTS$\cup\{(1024, \text{'Clark'}, \text{'CSE'}, 26)\}$

Delete

- Analogous to insert, but $-$ operator instead of $\cup$ operator.
- Can only delete whole tuples, cannot delete values of particular attributes.
- STUDENTS $\longleftarrow$ 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).