Advanced Management of Data

Self-Study!!! - The Relational Model

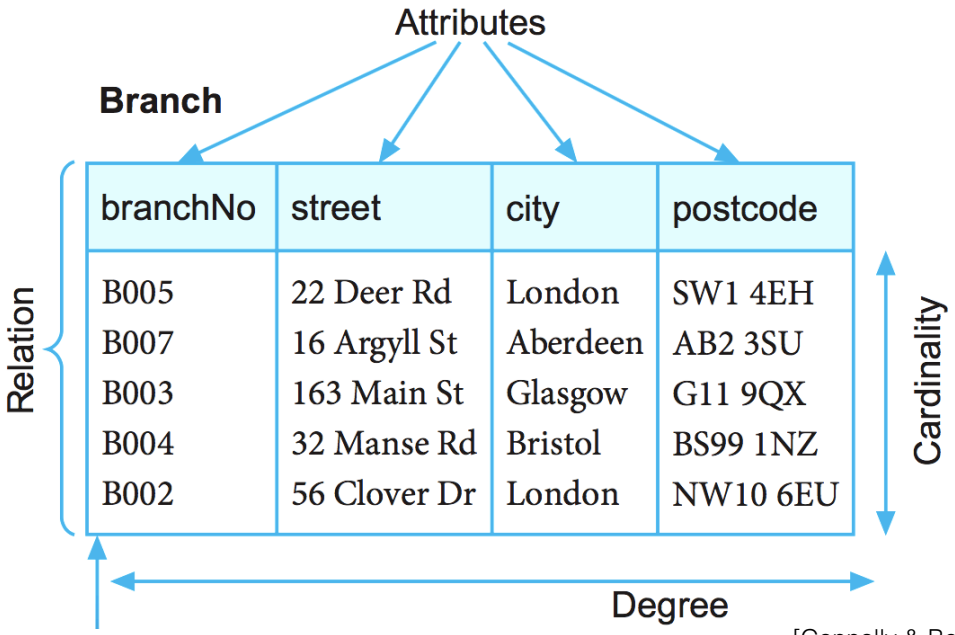
The Relational Model

- Relational Data Structure
- Keys & Integrity Constraints
- Relational Algebra & Basic SQL

Relational Data Structure

Terminology

- A **relation** is a table with columns and rows
- An **attribute** is a named column of a relation
- A **domain** is the set of allowable values for one or more attributes.
- A **tuple** is a row of a relation.



[Connolly & Begg]

- The **degree** of a relation is the number of attributes it contains.
- The **cardinality** of a relation is the number of tuples it contains.
- A **relational database** is a collection of normalized relations with distinct relation names.
  - normalized means, that each attribute of each tuple must have only a single value

Properties of Relations

- the relation has a name that is distinct from all other relation names in the relational schema
- each cell of the relation contains exactly one atomic (single) value
- each attribute (within a relation) has a distinct name
- the values of an attribute are all from the same domain
- each tuple is distinct; there are no duplicate tuples
- the order of attributes has no significance
- the order of tuples has no significance
- the name of a relation, the names and domains of its attributes are called relational schema

Relational Keys

Superkey

An attribute, or set of attributes, that uniquely identifies a tuple within a relation.

Candidate Key

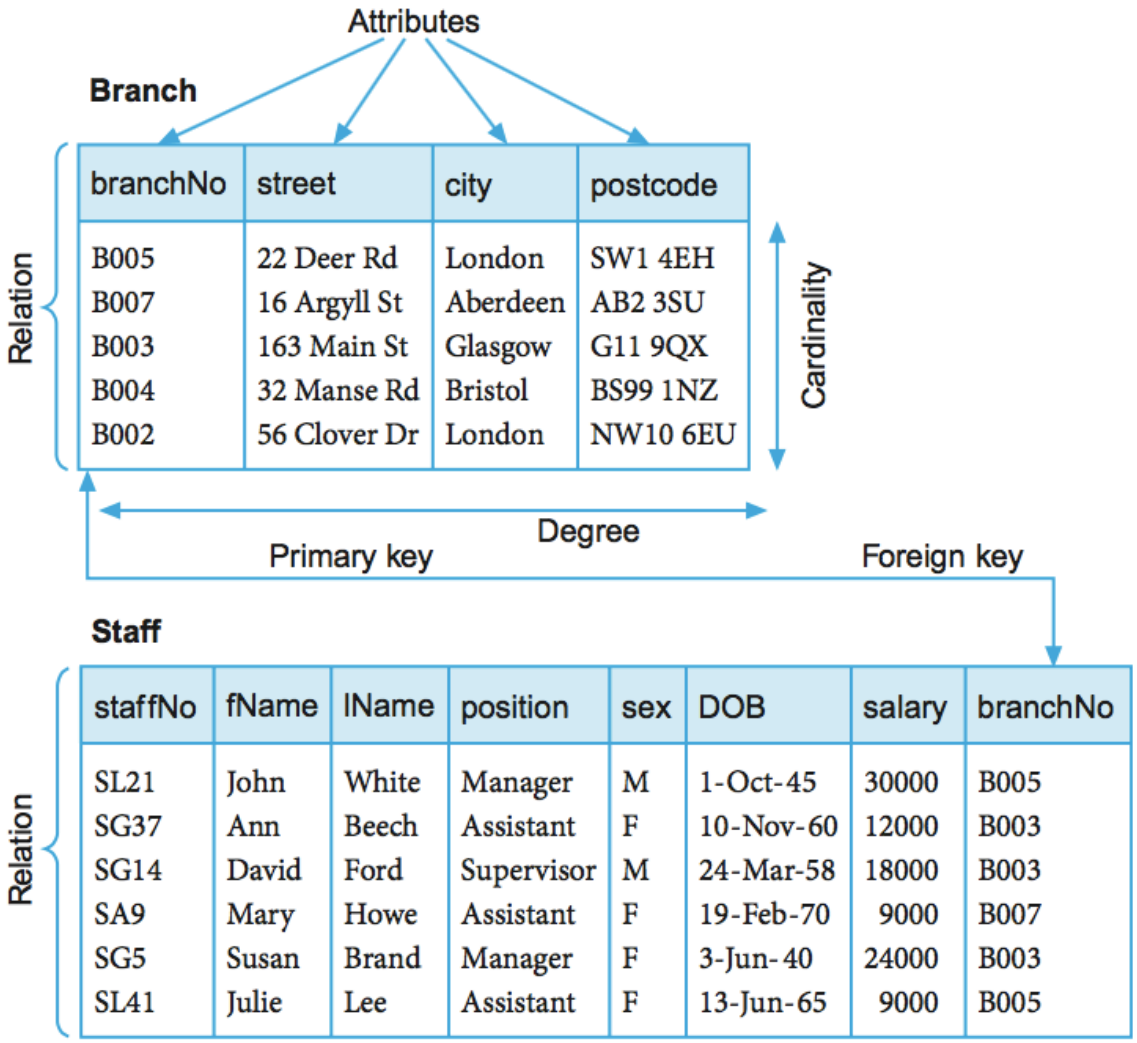
A superkey such that no proper subset is also a superkey within the relation.

Primary Key

The candidate key that is selected to identify tuples uniquely within the relation.

Foreign Key

An attribute, or set of attributes, within one relation that matches the candidate key of some (possibly the same) relation.



[Connolly & Begg]

Integrity Constraints

Null

Represents a value for an attribute that is currently unknown or is not applicable for this tuple.

Please note that value *null* is not the same as the numeric value *zero* or a text string filled with spaces. Therefore, nulls have to be treated differently from other values.

Domain integrity

Describes the restriction on the set of allowed values for an attribute.

Entity Integrity

No attribute of a primary key can be null.

Referential Integrity

If a foreign key exists in a relation, either the foreign key value must match a candidate key value of some tuple in its home relation or the foreign key value must be wholly null.

(Relational) Views

Base Relation

A named relation, whose tuples are physically stored in the database.

(Relational) View

The dynamic result of one or more relational operations operating on the base relations to produce another relation. A view is a *virtual relation* that does not necessarily exist in the database but can be produced upon request by a particular user, at the time of request.

Purpose of Views

- provide a powerful and flexible security mechanism by hiding parts of the database from certain users.
- permit users to access data in a way that is customized to their needs, so that the same data can be seen by different users in different ways
- can simplify complex operations on the base relations.

Relational Algebra

Relational Algebra

The relational algebra is a theoretical language with operations that work on one or more relations to define another relation without changing the original relation(s).

Closure

Since both the operands and the results of relational operations are relations, the output from one operation can become the input to another operation. Hence, expressions can be nested and relations are closed under the Algebra.

Operations

- Five fundamental operations perform most of relevant retrieval operations:
  Selection, Projection, Cartesian product, Union, Set difference
- Additionally, there are some convenience operations, which can be expressed in terms of the fundamental operations, e.g. Join, Intersection, and Division operations
- Actually, in some situations we need an additional fundamental operation, in order to rename relational components to resolve ambiguities.

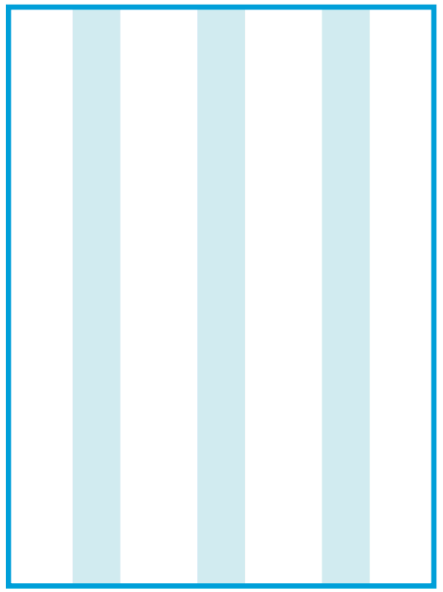
SQL

- SQL (Structured Query Language) is an easy-to-use nonprocedural language
- SQL is both the formal and de facto standard language for defining and manipulating relational databases.
- The select statement is the most important statement in the language and is used to express a query. It combines three fundamental relational algebra operations of projection, selection and cartesian product / join.
- the select statement derives a result table from a base table or a number of base tables.
- tables are technical representations of relations, but show a few differences:
  - there is a order of tuples in a table, which is the basis of sorting
  - a table can contain duplicates, which are not automatically deleted after an operation
  - relations can have an infinite number of tuples, the number of tuples of a table is limited

Unary Operations (1)

Projection

$$\Pi_{a_1 \ldots, a_n}(R)$$



The Projection operation works on a single relation R.

It defines a relation that contains a vertical subset of R, extracting the values of specified attributes and **eliminating duplicates**.

Projection in SQL

`select distinct` $a_1,\ldots,a_n$ `from` **R**

Note that duplicates would not be eliminated without keyword `distinct` in SQL.

π - Examples

R				
a	b	c	d	e
1	a1	45	xx	5
2	a1	23	xx	5
3	a2	13	xx	5
4	b1	99	xx	7
5	b1	10	xx	7

$\pi_{a,b,c}(R)$		
a	b	c
1	a1	45
2	a1	23
3	a2	13
4	b1	99
5	b1	10

```
select a,b,c
from R
```

$\pi_{b,e}(R)$	
b	e
a1	5
a2	5
b1	7

```
select distinct b,e
from R
```

$\pi_{d}(R)$
d
xx

```
select distinct d
from R
```

To output the entire relation R, we can write

```
select * from R
```

Unary Operations (2)

Selection

$$\sigma_{predicate}(R)$$

The Selection operation works on a single relation **R.**

It defines a relation that contains a horizontal subset of R, containing only those tuples of **R** that satisfy the specified condition *(**predicate**).*

Selection in SQL

```
select * from R where predicate
```

σ - Examples

$\sigma_{e=5}(R)$				
a	b	c	d	e
1	a1	45	xx	5
2	a1	23	xx	5
3	a2	13	xx	5

```
select * from R where e=5
```

If no tuple satisfies the predicate, a selection operation results in the relational schema of R without tuples.

$\sigma_{b='a3'}(R)$				
a	b	c	d	e

```
select * from R where b='a3'
```

R				
a	b	c	d	e
1	a1	45	xx	5
2	a1	23	xx	5
3	a2	13	xx	5
4	b1	99	xx	7
5	b1	10	xx	7

$\sigma_{c>40 \wedge e<7}(R)$				
a	b	c	d	e
1	a1	45	xx	5

```
select * from R
where c>40 and e<7
```

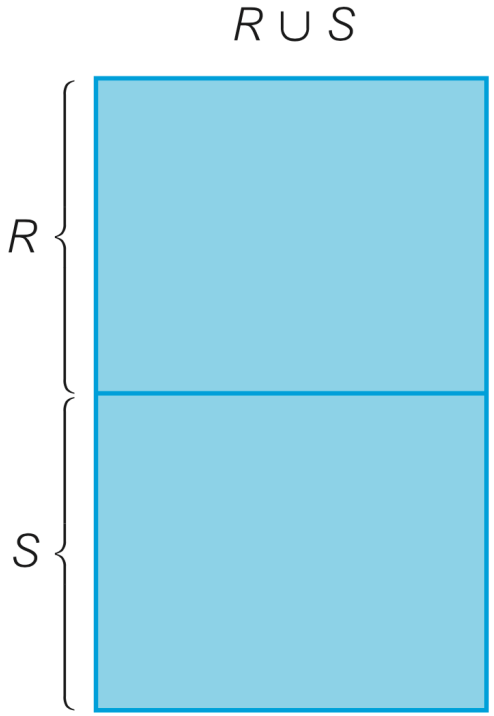
Set Operations (1)

Union

$$R \cup S$$

The union of two relations R and S defines a relation that contains all the tuples of R, or S, or both R and S, duplicate tuples being eliminated.

R and S must be union-compatible.



Union in SQL

```
select * from R union select * from S
```

Note that in SQL duplicates are also eliminated automatically.

∪ - Example (1)

R				
a	b	c	d	e
1	a1	45	xx	5
2	a1	23	xx	5
3	a2	13	xx	5
4	b1	99	xx	7
5	b1	10	xx	7

S				
a	b	c	d	e
6	b2	45	xx	7
7	b2	23	yy	9
8	c1	13	xx	9
9	c2	99	yy	9

R ∪ S				
a	b	c	d	e
1	a1	45	xx	5
2	a1	23	xx	5
3	a2	13	xx	5
4	b1	99	xx	7
5	b1	10	xx	7
6	b2	45	xx	7
7	b2	23	yy	9
8	c1	13	xx	9
9	c2	99	yy	9

∪ - Example (2)

R				
a	b	c	d	e
1	a1	45	xx	5
2	a1	23	xx	5
3	a2	13	xx	5
4	b1	99	xx	7
5	b1	10	xx	7

S				
a	b	c	d	e
5	b1	10	xx	7
7	b2	23	yy	9
3	a2	13	xx	5
9	c2	99	yy	9

R ∪ S				
a	b	c	d	e
1	a1	45	xx	5
2	a1	23	xx	5
3	a2	13	xx	5
4	b1	99	xx	7
5	b1	10	xx	7
7	b2	23	yy	9
9	c2	99	yy	9

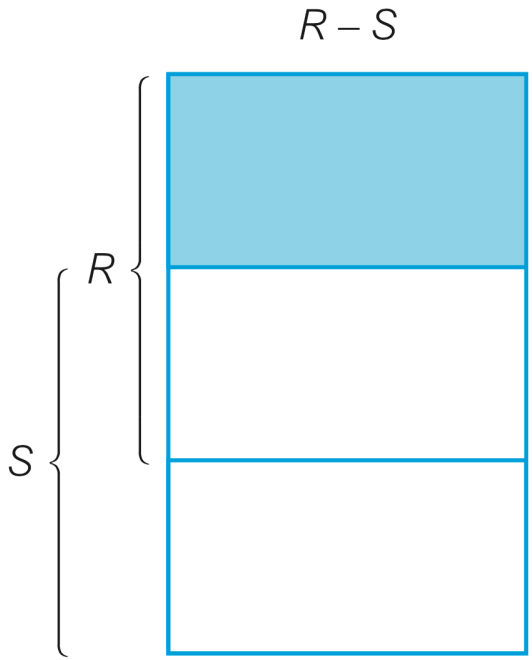
Set Operations (2)

Set Difference

R – S

The Set difference operation defines a relation consisting of the tuples that are in relation R, but not in S.

R and S must be union-compatible.



Set Difference in SQL

```
select * from R except select * from S
```

Set Difference - Example

R				
a	b	c	d	e
1	a1	45	xx	5
2	a1	23	xx	5
3	a2	13	xx	5
4	b1	99	xx	7
5	b1	10	xx	7

S				
a	b	c	d	e
5	b1	10	xx	7
7	b2	23	yy	9
3	a2	13	xx	5
9	c2	99	yy	9

R–S				
a	b	c	d	e
1	a1	45	xx	5
2	a1	23	xx	5
4	b1	99	xx	7

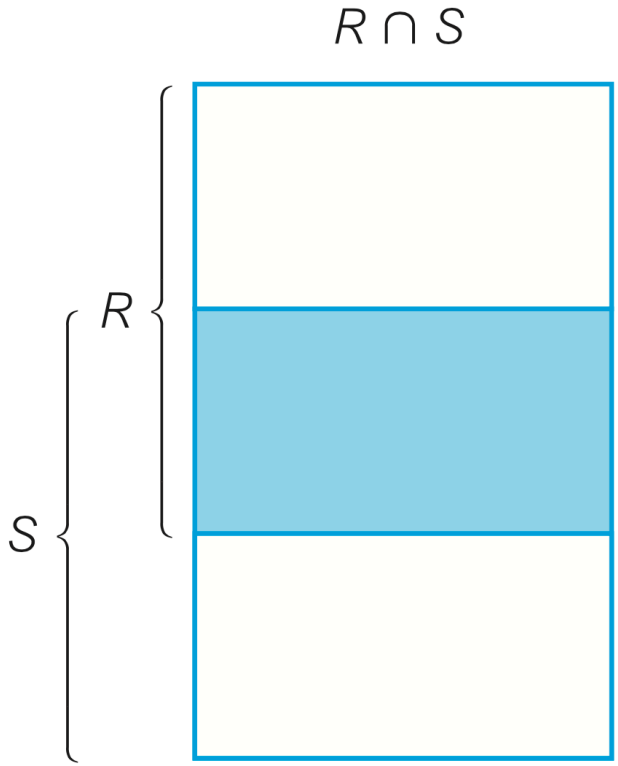
Set Operations (3)

Intersection

$$R \cap S$$

The Intersection operation defines a relation consisting of the set of all tuples that are in both R and S.

R and S must be union-compatible.



Intersection in SQL

```
select * from R
intersect
select * from S
```

R∩S = R–(R–S) - Example

R				
a	b	c	d	e
1	a1	45	xx	5
2	a1	23	xx	5
3	a2	13	xx	5
4	b1	99	xx	7
5	b1	10	xx	7

S				
a	b	c	d	e
5	b1	10	xx	7
7	b2	23	yy	9
3	a2	13	xx	5
9	c2	99	yy	9

R–S				
a	b	c	d	e
1	a1	45	xx	5
2	a1	23	xx	5
4	b1	99	xx	7

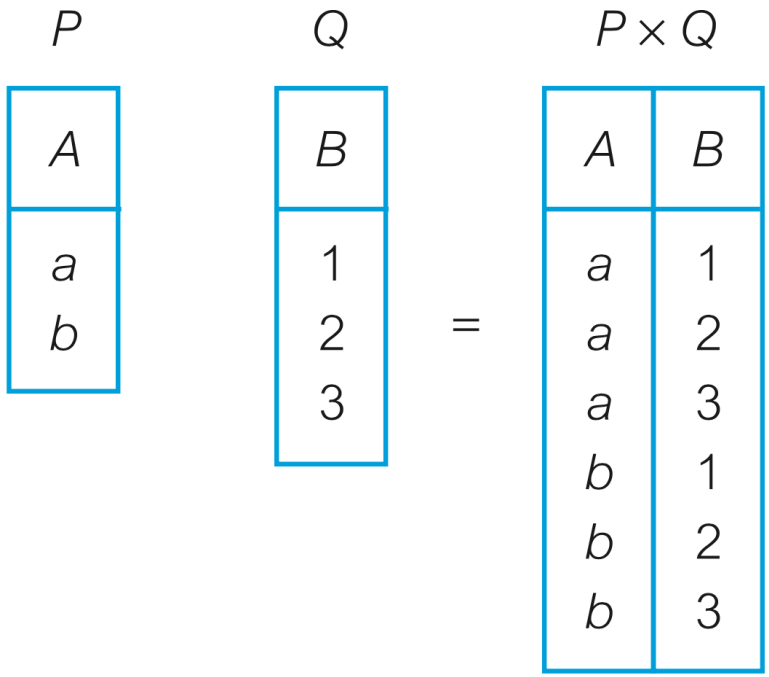
R–(R–S) = R∩S				
a	b	c	d	e
3	a2	13	xx	5
5	b1	10	xx	7

Set Operations (4)

Cartesian Product

$$\mathbf{R \times S}$$

The Cartesian product operation defines a relation that is the concatenation of every tuple of relation R with every tuple of relation S.



Cartesian Product in SQL

```
select * from R,S
```

× - Example

R		
a	b	c
1	a1	45
2	a1	23
5	b1	10

S			
f	g	h	k
6	b2	45	xx
7	b2	23	yy
8	c1	13	xx
9	c2	99	yy

R×S						
a	b	c	f	g	h	k
1	a1	45	6	b2	45	xx
1	a1	45	7	b2	23	yy
1	a1	45	8	c1	13	xx
1	a1	45	9	c2	99	yy
2	a1	23	6	b2	45	xx
2	a1	23	7	b2	23	yy
2	a1	23	8	c1	13	xx
2	a1	23	9	c2	99	yy
5	b1	10	6	b2	45	xx
5	b1	10	7	b2	23	yy
5	b1	10	8	c1	13	xx
5	b1	10	9	c2	99	yy

The Rename Operation

The relational algebra operations can be of arbitrary complexity. We can decompose such operations into a series of smaller relational algebra operations and give a name to the results of intermediate expressions.

Additionally, sometimes we need to resolve ambiguities when nesting relations after certain operations.

Renaming

$$\rho_S(E) \text{ or}$$
$$\rho_{S(a_1, a_2, \ldots, a_n)}(E)$$

The Rename operation provides a new name **S** for the expression **E**, and optionally names the attributes as $a_1, a_2, \ldots, a_n$.

Renaming in SQL

```
select e1 as a1, e2 as a2, …, en as an from E S
```

ρ - Example

R				
a	b	c	d	e
1	a1	45	xx	5
2	a1	23	xx	5
3	a2	13	xx	5
4	b1	99	xx	7
5	b1	10	xx	7

S				
f	g	h	m	n
1	a1	45	xx	5
2	a1	23	xx	5
3	a2	13	xx	5
4	b1	99	xx	7
5	b1	10	xx	7

```
select a as f, b as g, c as h, d as m, e as n from R S
```

Qualification of Attributes

R				
a	b	c	d	e
1	a1	45	xx	5
2	a1	23	xx	5

S				
a	b	c	d	e
5	b1	10	xx	7
7	b2	23	yy	9

R×S									
a	b	c	d	e	a	b	c	d	e
1	a1	45	xx	5	5	b1	10	xx	7
1	a1	45	xx	5	7	b2	23	yy	9
2	a1	23	xx	5	5	b1	10	xx	7
2	a1	23	xx	5	7	b2	23	yy	9

$\sigma_{R.d = S.d}(R \times S)$									
R.a	R.b	R.c	R.d	R.e	S.a	S.b	S.c	S.d	S.e
1	a1	45	xx	5	5	b1	10	xx	7
2	a1	23	xx	5	5	b1	10	xx	7

Nested Operations Example

Query

What are the names of lectures that are attended by a student with matriculation number 134534?

Relational Algebra

π[Name] (σ[MatrNo=134534 ∧ Lecture.LectNo=Attendance.LectNo] (Lecture×Attendance))

SQL

```
select Name
from   Lecture, Attendance
where  MatrNo=134534 and Lecture.LectNo=Attendance.LectNo
```

Lecture	
LectNo	Name
123	Data Structures
234	Databases
345	Algorithms

Attendance	
LectNo	MatrNo
123	134534
234	123456
345	234876
234	134534

Example (2)

π[Name] (σ[MatrNo=134534 ∧ Lecture.LectNo=Attendance.LectNo] **(Lecture×Attendance)**)

Lecture	
LectNo	Name
123	Data Structures
234	Databases
345	Algorithms

Attendance	
LectNo	MatrNo
123	134534
234	123456
345	234876
234	134534

Lecture×Attendance			
LectNo	Name	LectNo	MatrNo
123	**Data Structures**	**123**	**134534**
123	**Data Structures**	**234**	**123456**
123	**Data Structures**	**345**	**234876**
123	**Data Structures**	**234**	**134534**
234	**Databases**	**123**	**134534**
234	**Databases**	**234**	**123456**
234	**Databases**	**345**	**234876**
234	**Databases**	**234**	**134534**
345	**Algorithms**	**123**	**134534**
345	**Algorithms**	**234**	**123456**
345	**Algorithms**	**345**	**234876**
345	**Algorithms**	**234**	**134534**

Example (3)

π[Name] (σ[MatrNo=134534 ∧ **Lecture.LectNo=Attendance.LectNo**] (Lecture×Attendance))

Lecture	
LectNo	Name
123	Data Structures
234	Databases
345	Algorithms

Attendance	
LectNo	MatrNo
123	134534
234	123456
345	234876
234	134534

Lecture×Attendance			
LectNo	Name	LectNo	MatrNo
123	**Data Structures**	**123**	**134534**
123	Data Structures	234	123456
123	Data Structures	345	234876
123	Data Structures	234	134534
234	Databases	123	134534
234	**Databases**	**234**	**123456**
234	Databases	345	234876
234	**Databases**	**234**	**134534**
345	Algorithms	123	134534
345	Algorithms	234	123456
345	**Algorithms**	**345**	**234876**
345	Algorithms	234	134534

Example (4)

π[Name] (**σ[MatrNo=134534 ∧ Lecture.LectNo=Attendance.LectNo]** (Lecture×Attendance))

Lecture×Attendance			
LectNo	Name	LectNo	MatrNo
123	**Data Structures**	**123**	**134534**
123	Data Structures	234	123456
123	Data Structures	345	234876
123	Data Structures	234	134534
234	Databases	123	134534
234	Databases	234	123456
234	Databases	345	234876
234	**Databases**	**234**	**134534**
345	Algorithms	123	134534
345	Algorithms	234	123456
345	Algorithms	345	234876
345	Algorithms	234	134534

Example (5)

π[Name] (σ[MatrNo=134534 ∧ Lecture.LectNo=Attendance.LectNo] (Lecture×Attendance))

σ[MatrNo=134534 ∧ Lecture.LectNo=Attendance.LectNo] (Lecture×Attendance)			
LectNo	Name	LectNo	MatrNo
123	Data Structures	123	134534
234	Databases	234	134534

Resulting Relation
Name
Data Structures
Databases

What are the names of lectures that are attended by a student with matriculation number 134534?

Join Operations (1)

Theta-Join

$$\mathbf{R} \bowtie_{F} \mathbf{S}$$

The Theta join operation defines a relation that contains tuples satisfying the predicate *F* from the Cartesian product of **R** and **S**.

The predicate ***F*** is of the form $\mathbf{R.a_i\, u\, S.b_i}$, where **u** may be one comparison operator.

We can rewrite the Theta join in terms of the basic Selection and Cartesian product operations:

$$R \bowtie_{F} S = \sigma_{F}(R \times S)$$

Equijoin

If the predicate ***F*** of a Theta-Join contains only equality, we use the term **Equijoin**.

Example - Equijoin

R				
a	b	c	d	e
1	a1	45	xx	5
2	a1	23	xx	5

S				
a	b	c	d	e
5	b1	10	xx	7
7	b2	23	yy	9

R×S									
a	b	c	d	e	a	b	c	d	e
1	a1	45	xx	5	5	b1	10	xx	7
1	a1	45	xx	5	7	b2	23	yy	9
2	a1	23	xx	5	5	b1	10	xx	7
2	a1	23	xx	5	7	b2	23	yy	9

$R \bowtie_{R.d = S.d} S$									
a	b	c	**d**	e	a	b	c	**d**	e
1	a1	45	**xx**	5	5	b1	10	**xx**	7
2	a1	23	**xx**	5	5	b1	10	**xx**	7

Join Operations (2)

Natural Join

$$\mathbf{R} \bowtie \mathbf{S}$$

The Natural join is an Equijoin of the two relations R and S over all common attributes.

One occurrence of each common attribute is eliminated from the result.

T

A	*B*
a	1
b	2

U

B	*C*
1	*x*
1	*y*
3	*z*

$T \bowtie U$

A	*B*	*C*
a	1	*x*
a	1	*y*

Join Operations (3)

Outer Join

$$\mathbf{R} \bowtie \mathbf{S}$$

The **Left Outer Join** is a join in which tuples from **R** that do not have matching values in the common attributes of **S** are also included in the result relation. Missing values in the second relation are set to **null**.

Similarly, the **Right Outer join** keeps every tuple from **S** in the result.

The **Full Outer join** keeps all tuples in both relations, padding tuples with nulls when no matching tuples are found.

T

A	*B*
a	1
b	2

U

B	*C*
1	*x*
1	*y*
3	*z*

$T \bowtie_B U$

A	*B*	*C*
a	1	*x*
a	1	*y*
b	2	

Join Operations (4)

Semijoin

$$\mathbf{R} \triangleright_{F} \mathbf{S}$$

The Semijoin operation defines a relation that contains the tuples of **R** that participate in the join of **R** with **S** satisfying the predicate ***F***.

We can rewrite the Semijoin using the Projection and Join operations:

$$R \triangleright_{F} S = \Pi_{A}(R \bowtie_{F} S)$$

(*A* is the set of all attributes for R)

T

A	*B*
a	1
b	2

U

B	*C*
1	*x*
1	*y*
3	*z*

$T \triangleright_{B} U$

A	*B*
a	1

Division Operation

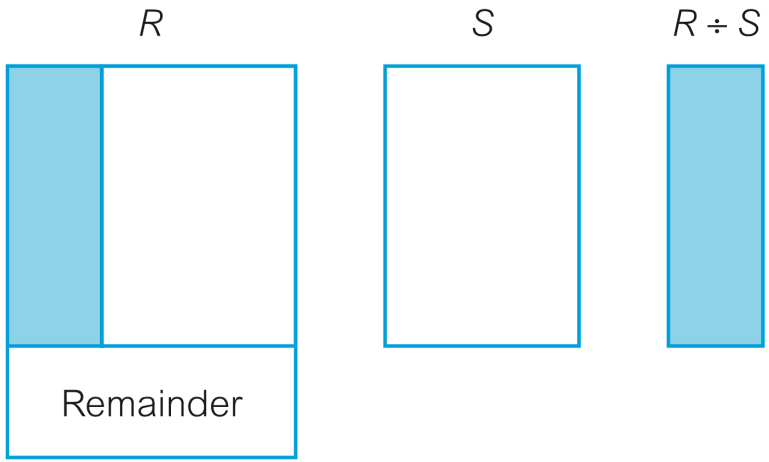
Division

R ÷ S

Assume that relation R is defined over the attribute set A and relation S is defined over the attribute set B such that B is a subset of A.

Let C = A – B, that is, C is the set of attributes of R that are not attributes of S.

The Division operation defines a relation over the attributes C that consists of the set of tuples from R that match the combination of **every** tuple in S.



V

A	*B*
a	1
a	2
b	1
b	2
c	1

W

B
1
2

V ÷ *W*

A
a
b