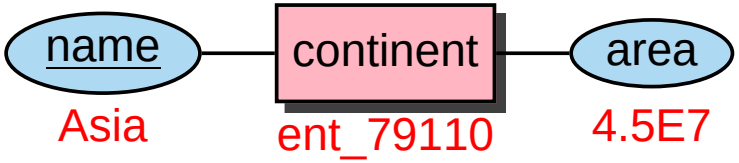
ENTITY TYPES

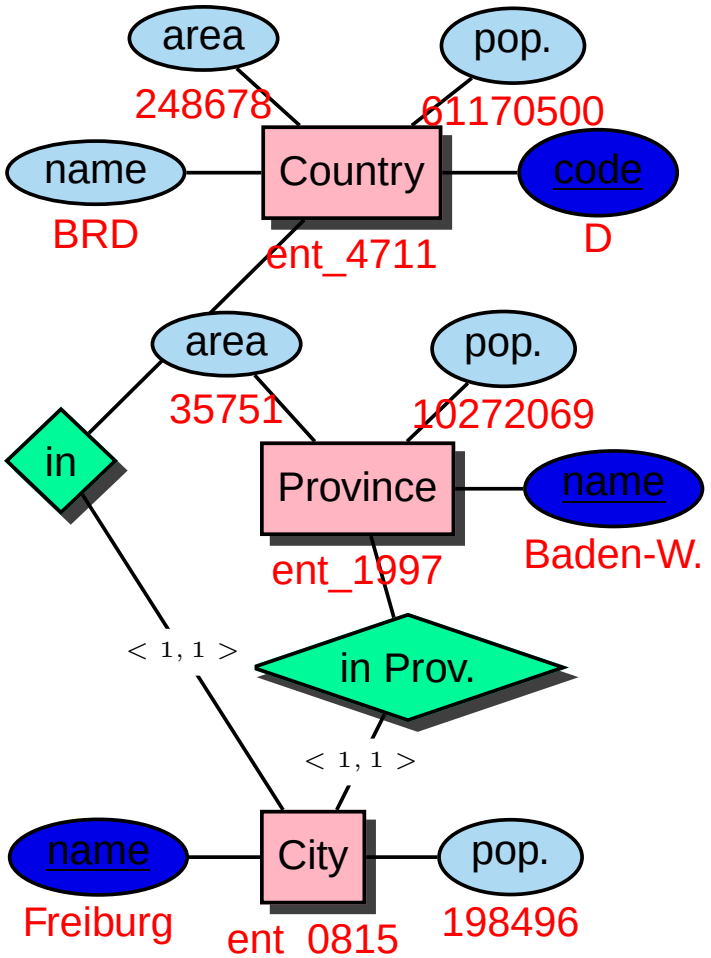
$(E_{ER}, \{A_1, \ldots, A_n\}) \rightarrow E(A_1, \ldots, A_n)$



Continent	
Name	**Area**
VARCHAR(20)	NUMBER
Europe	9562489.6
Africa	3.02547e+07
Asia	4.50953e+07
America	3.9872e+07
Australia	8503474.56

Weak Entity Types

For weak entity types, the key attributes of the identifying entity type(s) must be added.



City				
Name	**Country**	**Province**	**Population**	**...**
Freiburg	D	Baden-W.	198496	..
Berlin	D	Berlin	3472009	..
..	..	..	..	..

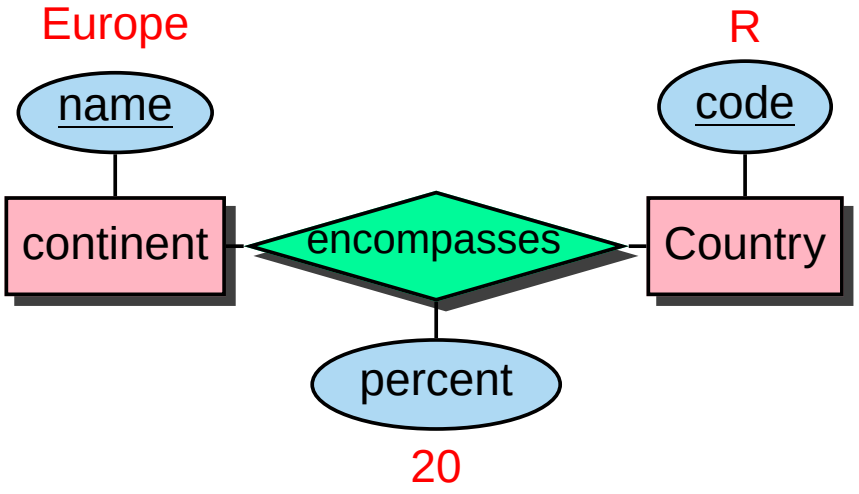
Relationship Types

$(B_{ER}, \{RO_1 : E_1, \ldots, RO_k : E_k\}, \{A_1, \ldots, A_m\}) \rightarrow$

$B(E_1\_K_{11}, \ldots, E_1\_K_{1p_1}, \ldots, E_k\_K_{k1}, \ldots, E_k\_K_{kp_k}, A_1, \ldots, A_m)$

where $\{K_{i1}, \ldots, K_{ip_i}\}$ are the primary keys of $E_i, 1 \leq i \leq k$.

(it is allowed to rename, e.g., to use *Country* for *Country.Code*)

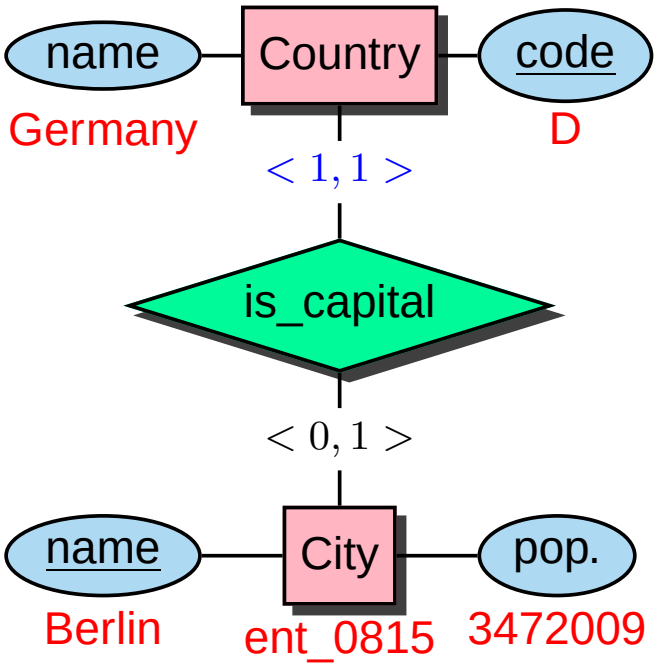


encompasses		
Country	**Continent**	**Percent**
VARCHAR(4)	VARCHAR(20)	NUMBER
R	Europe	20
R	Asia	80
D	Europe	100
...	...	...

- Note: for weak entity types, the global key must be used (exercise: is_capital)

Relationship Types: 1:n-Relationships

In case that $k = 2$ (binary relationship) and a (0,1)- or (1,1)-relationship complexity (i.e., n:1-relations), the relation schema of the relationship type and that of the entity type can be merged (into the relation schema for the entity type)

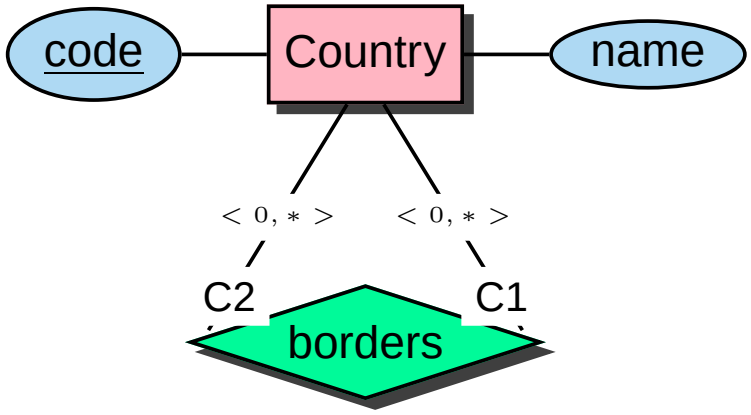


Country					
Name	**code**	**Population**	**Capital**	**Province**	**...**
Germany	D	83536115	Berlin	Berlin	..
Austria	A	8023244	Vienna	Vienna	..
Canada	CDN	28820671	Ottawa	Quebec	..
Bolivia	BOL	7165257	La Paz	Bolivia	..
..	..	..	..	..	..

Other examples: flows_into, headquarters of organizations

Relationship Types

In case that for some relationship type, the keys of involved entity types have coinciding names, the role specifications may be used to guarantee the uniqueness of key attributes in the relationship type.



borders	
Country1	**Country2**
D	F
D	CH
CH	F
..	..

Relational Algebra

- Recall, the Relational Model consists of the elements: relations, which are made up of attributes.
- A relation is a set of attributes with values for each attribute such that:
    1. Each attribute value must be a single value only (atomic).
    2. All values for a given attribute must be of the same type (or domain).
    3. Each attribute name must be unique.
    4. The order of attributes is insignificant
    5. No two rows (tuples) in a relation can be identical.
    6. The order of the rows (tuples) is insignificant.
- Relational Algebra is a collection of operations on Relations.
- Relations are operands and the result of an operation is another relation.
- Two main collections of relational operators:
    1. Set theory operations:
       Union, Intersection, Difference and Cartesian product.
    2. Specific Relational Operations:
       Selection, Projection, Join, Division

Set Theoretic Operations

Consider the following relations **R** and **S**

R

First	Last	Age
Bill	Smith	22
Sally	Green	28
Mary	Keen	23
Tony	Jones	32

S

First	Last	Age
Forrest	Gump	36
Sally	Green	28
DonJuan	DeMarco	27

- Union: R ∪S
  Result: Relation with tuples from R and S with duplicates removed.
- Difference: R - S
  Result: Relation with tuples from R but not from S
- Intersection: R ∩S
  Result: Relation with tuples that appear in both R and S.

R ∪S

First	Last	Age
Bill	Smith	22
Sally	Green	28
Mary	Keen	23
Tony	Jones	32
Forrest	Gump	36
DonJuan	DeMarco	27

R - S

First	Last	Age
Bill	Smith	22
Mary	Keen	23
Tony	Jones	32

R ∩S

First	Last	Age
Sally	Green	28

Cartesian Product

- Produce all combinations of tuples from two relations.

  R

First	Last	Age
Bill	Smith	22
Mary	Keen	23
Tony	Jones	32

  S

Dinner	Dessert
Steak	Ice Cream
Lobster	Cheesecake

  R X S

First	Last	Age	Dinner	Dessert
Bill	Smith	22	Steak	Ice Cream
Bill	Smith	22	Lobster	Cheesecake
Mary	Keen	23	Steak	Ice Cream
Mary	Keen	23	Lobster	Cheesecake
Tony	Jones	32	Steak	Ice Cream
Tony	Jones	32	Lobster	Cheesecake

Selection Operator

- Selection and Projection are *unary* operators.
- The selection operator is sigma: σ
- The selection operation acts like a *filter* on a relation by returning only a certain number of tuples.
- The resulting relation will have the same degree as the original relation.
- The resulting relation may have fewer tuples than the original relation.
- The tuples to be returned are dependent on a *condition* that is part of the selection operator.
- $\sigma_C(R)$ Returns only those tuples in R that satisfy condition **C**
- A condition C can be made up of any combination of comparison or logical operators that operate on the attributes of R.
    - Comparison operators:

      $=$ $<$ $>$ $\geq$ $\leq$ $\neq$
    - Logical operators: $\wedge$ $\vee$ $\neg$
- Use the Truth tables (memorize these) for logical expressions:

$\wedge$	**T**	**F**
T	T	F
F	F	F

$\vee$	**T**	**F**
T	T	T
F	T	F

$\neg$	**T**	**F**
	F	T

Selection Examples

Assume the following relation EMP has the following tuples:

Name	Office	Dept	Rank
Smith	400	CS	Assistant
Jones	220	Econ	Adjunct
Green	160	Econ	Assistant
Brown	420	CS	Associate
Smith	500	Fin	Associate

- Select only those Employees in the CS department:

  $\sigma_{\text{Dept = 'CS'}}$ (EMP)

  Result:

Name	Office	Dept	Rank
Smith	400	CS	Assistant
Brown	420	CS	Associate

- Select only those Employees with last name Smith who are assistant professors:
  $\sigma_{\text{Name = 'Smith'} \wedge \text{Rank = 'Assistant'}}(\text{EMP})$
  Result:

Name	Office	Dept	Rank
Smith	400	CS	Assistant

- Select only those Employees who are either Assistant Professors or in the Economics department:
  $\sigma_{\text{Rank = 'Assistant'} \vee \text{Dept = 'Econ'}}(\text{EMP})$
  Result:

Name	Office	Dept	Rank
Smith	400	CS	Assistant
Jones	220	Econ	Adjunct
Green	160	Econ	Assistant

- Select only those Employees who are not in the CS department or Adjuncts:
  $\sigma_{\neg (Rank = 'Adjunct' \vee Dept = 'CS')} (EMP)$
  Result:

Name	Office	Dept	Rank
Green	160	Econ	Assistant
Smith	500	Fin	Associate

Projection Operator

- Projection is also a Unary operator.
- The Projection operator is pi: π
- Projection limits the *attributes* that will be returned from the original relation.
- The general syntax is: $\pi_{attributes}$ R
  Where *attributes* is the list of attributes to be displayed and R is the relation.
- The resulting relation will have the same number of tuples as the original relation (unless there are duplicate tuples produced).
- The degree of the resulting relation may be equal to or less than that of the original relation.

- Project only the names and departments of the employees:
  $\pi_{\text{name, dept}}$ (EMP)
  Results:

Name	Dept
Smith	CS
Jones	Econ
Green	Econ
Brown	CS
Smith	Fin

Combining Selection and Projection

- The selection and projection operators can be combined to perform both operations.
- Show the names of all employees working in the CS department:
  π_{name} ($\sigma_{\text{Dept = 'CS'}}$ (EMP))
  Results:

Name
Smith
Brown

- Show the name and rank of those Employees who are not in the CS department or Adjuncts:

$$\pi_{\text{name, rank}} (\sigma \neg (\text{Rank} = \text{'Adjunct'} \vee \text{Dept} = \text{'CS'}) (\text{EMP}))$$

Result:

Name	Rank
Green	Assistant
Smith	Associate

Aggregate Functions

- We can also apply *Aggregate functions* to attributes and tuples:
  - SUM
  - MINIMUM
  - MAXIMUM
  - AVERAGE, MEAN, MEDIAN
  - COUNT

Name	Office	Dept	Salary
Smith	400	CS	45000
Jones	220	Econ	35000
Green	160	Econ	50000
Brown	420	CS	65000
Smith	500	Fin	60000

- Find the minimum Salary: $\Im_{\text{MIN (salary)}}$ (EMP)
  Results:

MIN(salary)
35000

- Find the average Salary: $\Im_{\text{AVG (salary)}}$ (EMP)
  Results:

AVG(salary)
51000

- Count the number of employees in the CS department: $\Im_{\text{COUNT (name)}}$ ($\sigma_{\text{Dept = 'CS'}}$ (EMP))
  Results:

COUNT(name)
2

- Find the total payroll for the Economics department: $\Im_{\text{SUM (salary)}}$ ($\sigma_{\text{Dept = 'Econ'}}$ (EMP))
  Results:

SUM(salary)
85000

Join Operation

- Join operations bring together two relations and combine their attributes and tuples in a specific fashion.
- The generic join operator (called the *Theta Join* is: $\bowtie$
- It takes as arguments the attributes from the two relations that are to be joined.
- For example assume we have the EMP relation as above and a separate DEPART relation with (Dept, MainOffice, Phone) :
  EMP $\bowtie_{\text{EMP.Dept = DEPART.Dept}}$ DEPART

- The join condition can be $=$ $<$ $>$ $\geq$ $\leq$ $\neq$
- When the join condition operator is $=$ then we call this an *Equijoin*
- Note that the attributes in common are repeated.

Join Examples

Assume we have the EMP relation from above and the following DEPART relation:

Dept	MainOffice	Phone
CS	404	555-1212
Econ	200	555-1234
Fin	501	555-4321
Hist	100	555-9876

- Find all information on every employee including their department info:
  EMP $\bowtie_{emp.Dept = depart.Dept}$ DEPART
  Results:

Name	Office	EMP.Dept	Salary	DEPART.Dept	MainOffice	Phone
Smith	400	CS	45000	CS	404	555-1212
Jones	220	Econ	35000	Econ	200	555-1234
Green	160	Econ	50000	Econ	200	555-1234
Brown	420	CS	65000	CS	404	555-1212
Smith	500	Fin	60000	Fin	501	555-4321

Natural Join

- Notice in the generic (Theta) join operation, any attributes in common (such as `dept` above) are repeated.
- The *Natural Join* operation removes these duplicate attributes.
- The natural join operator is: *****
- We can also assume using ***** that the join condition will be = on the two attributes in common.
- Example: EMP ***** DEPART
  Results:

Name	Office	Dept	Salary	MainOffice	Phone
Smith	400	CS	45000	404	555-1212
Jones	220	Econ	35000	200	555-1234
Green	160	Econ	50000	200	555-1234
Brown	420	CS	65000	404	555-1212
Smith	500	Fin	60000	501	555-4321

Underlying Data Model

SQL uses the relational model:

- SQL relations are **multisets (bags)** of tuples (i.e., they can contain duplicates)
- Notions: Relation $\rightsquigarrow$ Table
  Tuple $\rightsquigarrow$ Row
  Attribute $\rightsquigarrow$ Column

The relational algebra serves as theoretical base for SQL as a query language.

The Language

Basic structure of SQL queries:

`SELECT` $A_1, \ldots, A_n$	(. . . corresponds to π in the algebra)
`FROM` $R_1, \ldots, R_m$	(. . . specifies the contributing relations)
`WHERE` F	(. . . corresponds to σ in the algebra)

corresponds to the algebra expression $\pi[A_1, \ldots, A_n](\sigma[F](r_1 \times \ldots \times r_m))$

- Note: cartesian product $\rightarrow$ aliasing

General Structure:

`SELECT` $A_1, \ldots, A_n$	list of attributes
`FROM` $R_1, \ldots, R_m$	list of relations
`WHERE` F	condition(s)
`GROUP BY` $B_1, \ldots, B_k$	list of grouping attributes
`HAVING` G	condtion on groups
`ORDER BY` H	sort order

Subqueries correspond to subtrees in the relational algebra.

Constructing Queries

For each problem there are multiple possible equivalent queries in SQL (cf. Example 4.15). The choice is a matter of personal taste.

- analyze the problem “systematically”:
  - collect all relations (in the FROM clause) that are needed
  - generate a suitable conjunctive WHERE clause

  $\Rightarrow$ leads to a single “broad” SFW query
  (cf. conjunctive queries, relational calculus)
- analyze the problem “top-down”:
  - take the relations that directly contribute to the result in the (outer) FROM clause
  - do all further work in correlated subquery/-queries in the WHERE clause

  $\Rightarrow$ leads to a “main” part and nested subproblems
- decomposition of the problem into subproblems:
  - subproblems are solved by nested SFW queries that are combined in the FROM clause of a surrounding query

Projection

Example: The table **E** (for **EMPLOYEE**)

nr	name	salary
1	John	100
5	Sarah	300
7	Tom	100

SQL	Result	Relational algebra
`select salary from E`	**salary** 100 300	$\pi_{\text{salary}}(E)$
`select nr, salary from E`	**nr** **salary** 1 100 5 300 7 100	$\pi_{\text{nr, salary}}(E)$

Selection

The same table **E** (for **EMPLOYEE**) as above.

<table>
<tr><th>SQL</th><th>Result</th><th>Relational algebra</th></tr>
<tr><td><pre>select *
from E
where salary < 200</pre></td><td><table><tr><th>nr</th><th>name</th><th>salary</th></tr><tr><td>1</td><td>John</td><td>100</td></tr><tr><td>7</td><td>Tom</td><td>100</td></tr></table></td><td>$\sigma_{\text{salary} < 200}(E)$</td></tr>
<tr><td><pre>select *
from E
where salary < 200
and nr >= 7</pre></td><td><table><tr><th>nr</th><th>name</th><th>salary</th></tr><tr><td>7</td><td>Tom</td><td>100</td></tr></table></td><td>$\sigma_{\text{salary} < 200 \text{ and nr} >= 7}(E)$</td></tr>
</table>

Combination of projection and selection

<table>
<tr><th>SQL</th><th>Result</th><th>Relational algebra</th></tr>
<tr><td><pre>select name, salary
from E
where salary < 200</pre></td><td><table><tr><th>name</th><th>salary</th></tr><tr><td>John</td><td>100</td></tr><tr><td>Tom</td><td>100</td></tr></table></td><td>$\pi_{\text{name, salary}} (\sigma_{\text{salary} < 200}(E))$</td></tr>
</table>

Cartesian product

Example: The table **E** (for **EMPLOYEE**)

enr	ename	dept
1	Bill	A
2	Sarah	C
3	John	A

Example: The table **D** (for **DEPARTMENT**)

dnr	dname
A	Marketing
B	Sales
C	Legal

SQL	Result	Relational algebra

```
select *
from E, D
```

enr	ename	dept	dnr	dname
1	Bill	A	A	Marketing
1	Bill	A	B	Sales
1	Bill	A	C	Legal
2	Sarah	C	A	Marketing
2	Sarah	C	B	Sales
2	Sarah	C	C	Legal
3	John	A	A	Marketing
3	John	A	B	Sales
3	John	A	C	Legal

E **X** D