Data Model

- Set of concepts and constructs used to describe and organize data and their relationships
- Basic feature: structuring mechanism (also: type constructor) as in programming languages
- *Example*: in the relational DB model, ***relation*** constructor organizes data as sets of homogeneous (same type) records
- Two main types of data model:
  - ***Logical models***: used for organization of data at a level that abstracts from physical structures

    Examples: relational, network, hierarchical (traditional ones), object (more recent)

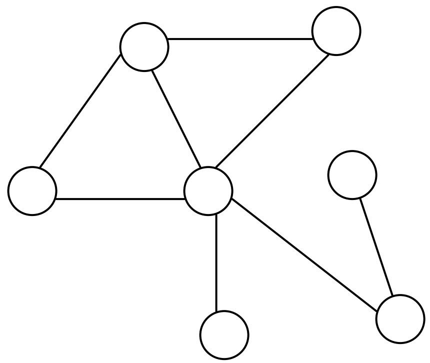
  - ***Conceptual models***: used to describe data in a way that is completely independent of any system, with the goal of representing the concepts of the real world; used in the early stages of DB design

    Most popular: Entity-Relationship model

Network Data Model

Characteristics:

- Data represented as collection of *records*
- Binary relationships represented as *links* (also called *sets,* and implemented as pointers)
- The model is represented by means of graph structures where:
    - o Nodes=records
    - o Edges=links



Relational Model

Characteristics:

- Data and relationships represented as values (relations)
- No explicit references, i.e., pointers as in the network model

=> higher level representation, while network model is closer to the physical structure of the DB

EXAMPLE: A Relational Database

STUDENTS

RegNum	Surname	FirstName	BirthDate
276545	Smith	Mary	25/11/1990
485745	Black	Anna	23/04/1991
200768	Verdi	Paolo	12/02/1991
587614	Smith	Lucy	10/10/1990
937653	Brown	Mavis	01/12/1990

EXAMS

Student	Grade	Course
276545	C	01
276545	B	04
937653	B	01
200768	B	04

COURSES

Code	Title	Tutor
01	Physics	Grant
03	Chemistry	Beale
04	Chemistry	Clark

EXAMPLE: A Network Database

STUDENTS

RegNum	Surname	FirstName	BirthDate
276545	Smith	Mary	25/11/1990
485745	Black	Anna	23/04/1991
200768	Verdi	Paolo	12/02/1991
587614	Smith	Lucy	10/10/1990
937653	Brown	Mavis	01/12/1990

EXAMS

Student	Grade	Course
	C	
	B	
	B	
	B	

COURSES

Code	Title	Tutor
01	Physics	Grant
03	Chemistry	Beale
04	Chemistry	Clark

Object-Oriented Data Model

Characteristics:

- Newer model: based on objects, classes, etc.
- Attributes: describe the state of an object
- Methods (also: actions) describe the behaviour of an object
- The object encapsulates both state and behaviour
- Development of OODBMS: still research topic (ODMG: Object Database Management Group)
- No universally agreed data model

Relational Model

- Proposed by E. F. Codd in 1970 in order to support data independence

- Used in almost all commercial DBMS since 1981

- It provides simple and declarative languages that are powerful and allow to express operations for access and manipulation of data

- It is based on the mathematical concept of **relation**;
  theoretical basis that allows to formally prove properties of data and operations

Relational Model

- **Relation:** subset of the **Cartesian product** of a list of **domains**

- **Domain**: a set (possibly infinite) of values;
  examples:
  - the set of integers is a domain;
  - the set of strings of characters with length=20 is a domain
  - {0,1} is a domain

- Let $D_1, D_2, \ldots D_k$ be domains. The Cartesian product of such domains, denoted by

$$D_1 \times D_2 \times \ldots \times D_k$$

is the set

$$\{(v_1, v_2, \ldots, v_k) \mid v_1 \in D_1, v_2 \in D_2, \ldots v_k \in D_k\}$$

Relational Model

- Example:

  let: $k = 2$, D_1 = {0,1}, and D_2 = {a,b,c}

  $D1 \times D_2$ = {(0,a), (0,b), (0,c), (1,a), (1,b), (1,c)}

- a **relation** is any subset of the Cartesian product of one or several domains. Example:

  {(0,a), (0,c),(1,b)} is a relation

  {(1,b), (1,c)} is a relation

- elements of a relation are called **tuples.**
  With reference to previous example (0,a), (0,c),(1,b), (1,c) are tuples

- a relation that is the subset of a Cartesian product of *k* domains is said to have **degree** *k*.
  With reference to previous example: relations have degree 2

Relational Model

- every tuple of a relation with degree k has k components. With reference to previous example: tuples have 2 components

- let r be a relation with degree k;
  - let t be a tuple of r
  - let i be an integer in $\{1,...,k\}$
  - $t[i]$ is the i-th component of t

Example: let $r = \{(0,a), (0,c),(1,b)\}$

$t = (0,a)$ is a tuple of r

$t[2] = a$

$t[1] = 0$

- the **cardinality** of a relation is the number of tuples belonging to the relation.

Example: relation $\{(0,a), (0,c),(1,b)\}$ has cardinality 3.

Relational Model

Alternative (simpler) definition

- A relation can be seen as a table in which each row is a tuple and each column corresponds to a component
- In this definition, columns have associated names, called **attribute names**
  - the pair (attribute name, domain) is called an **attribute**
- The set of attributes of a relation is called **schema**
- If a relation has name $\boldsymbol{R}$ and attribute names $A_1, A_2, \ldots, A_k$, the schema is often indicated by

  $\boldsymbol{R}(A_1, A_2, \ldots, A_k)$

- $U_R = \{A_1, A_2, \ldots, A_k\}$ is used to denote the set of all attribute names of relation $\boldsymbol{R}$

Example:

Relation ***Info_City***

City	*Region*	*Population*
Roma	Lazio	3,000,000
Milano	Lombardia	1,500,000
Genova	Liguria	800,000
Pisa	Toscana	150,000

schema ***Info_City***(*City, Region, Population*)

Relational Model

An alternative (simpler) definition

- in this definition of the relational model, components of tuples are indicated by attribute names (notation by name vs notation by position)

- let ***R***$(A_1, A_2, \ldots, A_k)$ be a relation schema, a tuple t on such a schema can be represented by the notation:

  $[A_1 : v_1,\ A_2 : v_2, \ldots, A_k : v_k]$ or by $(v1, v2, \ldots, vk)$

  where v_i is a value belonging to the domain of A_i (denoted $dom(A_i)$) for $i=1,\ldots,k$

  $t[A_i]$ indicates the value of the attribute named A_i of tuple t

- Example:

t = [City : Roma, Region : Lazio, Population : 3,000,000] <u>or</u> t = (Roma, Lazio, 3,000,000)

is a tuple defined on schema **Info_City**(City, Region, Population)

t[City] = Roma

Relational Value

Null Values

- sometimes no information is available on some components of entities represented in the DB

  i.e., no value is known for some attributes of some tuples

- special value (**null value**) denotes no value

  [often denoted '?']

Relational Model: Key

- The **key** of a relation is the <u>set of attributes</u> that uniquely identifies tuples of the relation

- More precisely, a set X of attributes of a relation R, is a *key* of R if it satisfies the following properties:

  1. for each status of R, no pair of distinct tuples *t'* and *t''* exist in R such that *t'* and *t''* have same value for all attributes in X;

  2. no proper subset (*) of X satisfies property (1).

- In the previous example:
  key(Info_City) = {City}
  there cannot be multiple cities with same name

  key(Info_City) = {City, Region}
  different cities with same name can exist but only in different regions

----

(*)S' is a proper subset of S, if it is a subset of S and S'$\neq$S.

Relational Model: Key

- A key cannot have null values
- There can be more than one set X in a relation that satisfies the two properties (several possible keys)
- Sometimes it is necessary to choose one key if the system does not support multiple keys.
- **Primary key** is the selected key
- A possible selection criterion is to choose the key most frequently used in queries
- Another criterion: choose the key with least number of attributes

Relational Model: Foreign Key

- Let *R* and *S* be two relations such that

  - *R* has a set of attributes *X*;

  - *S* has a set *Y* of attributes <u>as key</u>;

  Y is **foreign key** of *R* on *S* if *Y* is a subset of *X*

- In other words, if *R* has among its attributes a set *Y* of attributes that is key of a relation *S*, we say that *Y* is a **foreign** key of *R* on *S*

- *S* is said **referenced relation**

- Foreign keys allow to link tuples of different relations and provide a mechanism to model associations between entities

- A tuple *t* that references another tuple *t'* includes, among its attributes, one or more attributes whose value is the value of the key of *t'*

Relational Model: Example

We define two relations that contain information about employees of a company and the departments in which the company is organized

Employees (Emp#, Name, Job, Start_Date,Salary, Bonus, Dept#)

key(Employees) = {Emp#}

foreign-key(Employees) = {Dept#}

(referenced relation: Departments)

Departments(Dept#, Name_Dept, Office#, Division#, Manager)

key (Departments) = {Dept#}

Example

Employees

Emp#	Name	Job	Start_Date	Salary	Bonus	Dept#
7369	Rossi	engineer	17-Dec-90	1600,00	500,00	20
7499	Andrei	technician	20-Feb-91	800,00	?	30
7521	Bianchi	technician	20-Feb-91	800,00	100,00	30
7566	Rosi	manager	02-Apr-91	2975,00	?	20
7654	Martini	secretary	28-Sep-91	800,00	?	30
7698	Blacchi	manager	01-May-91	2850,00	?	30
7782	Neri	engineer	01-Jun-91	2450,00	200,00	10
7788	Scotti	secretary	09-Nov-91	800,00	?	20
7839	Dare	engineer	17-Nov-91	2600,00	300,00	10
7844	Turni	technician	08-Sep-91	1500,00	?	30
7876	Adami	engineer	28-Sep-91	1100,00	500,00	20
7900	Gianni	engineer	03-Dec-91	1950,00	?	30
7902	Fordi	secretary	03-Dec-91	1000,00	?	20
7934	Milli	engineer	23-Jan-92	1300,00	150,00	10
7977	Verdi	manager	10-Dec-90	3000,00	?	10

Departments

Dept#	Name_Dept	Office	Division	Manager
10	Civil Engineering	1100	D1	7977
20	R&D	2200	D1	7566
30	Surveying	5100	D2	7698

Relational Model: Referential Integrity Constraints

- imposed to guarantee that values refer to actual values in the referenced relation
- if a tuple *t* references $v_1,.....,v_n$ as values of a foreign key, there must be a tuple *t'* in the referenced relation with key values $v_1,.....,v_n$
- relations Employees and Departments verify this property
- consider the following tuple and assume it is inserted in relation Employees

  [Emp#: 7899, Name: Smith, Job: technician,
  Start_Date_A:03-Dec-91, Salary:2000,
  Bonus: 100, Dept#: 50]

  this tuple violates referential integrity as there is no department in relation Departments with Dept# = 50

- DB languages (SQL) allow the user to specify for which relations and attributes it is necessary to preserve referential integrity (and what to do when there is violation)

Query Languages for Relational DB

- Operations on DB:

  1. queries: read from the DB
  2. updates: change the content of the DB

- Both types of operations can be modeled as functions from DB to DB

- Formalization with reference to query languages:

  - *relational algebra*: a “procedural” language
  - *relational calculus*: a “declarative” language

- Later, we will see SQL: practical language for queries and updates

Operations in Relational Model

Two basic formalisms

1) **Relational Algebra:** queries are expressed by applying operators to relations

2) **Relational Calculus:** queries are expressed by means of logical formulas that must be satisfied by the tuples obtained as result of the query

Theoretical Result: the two formalisms have same expressive power (under certain assumptions).

Relational Algebra

- 5 basic operations:
  - *union*
  - *difference*
  - *Cartesian product*
  - *projection*
  - *selection*

- these operations completely define relational algebra

- every operation returns a relation as result; it is then possible to apply an operation to the result of another operation (closure property)

- there are additional operations that can be expressed in terms of the 5 basic operations

- these operations do not add expressive power to the set of basic operations but they are useful shortcuts and they are called *derived* operations

- the most important derived operation: *join*

- *renaming:* to modify names of attributes

Union

- Union of two relations R and S, indicated R $\cup$ S:

  set of tuples that are in R, or in S, or in both

- Union of two relations is possible only if the two relations have same degree; also: the first attribute of R must be compatible with the first attribute of S, the second attribute of R must be compatible with the second attribute of S and so on.

- if the two relations have different attribute names, in the returned relation by convention the names from the first relation (in this case R) are used, unless renaming is applied

- duplicate tuples are eliminated

- the degree of the returned relation is the same as the degree of the two original relations

Union

Example

A	B	C
a	b	c
d	a	f
c	b	d

relation R

D	E	F
b	g	a
d	a	f

relation S

A	B	C
a	b	c
d	a	f
c	b	d
b	g	a

$R \cup S$

Difference

- Difference of two relations R and S, indicated R – S:

  <u>set of tuples that are in R, but not in S</u>

- difference (like union) of two relations is possible only if the two relations have same degree and attributes are compatible

- if the two relations have different attribute names, in the returned relation by convention the names from the first relation (in this case R) are used, unless renaming is applied

- the degree of the returned relation is the same as the degree of the two original relations

Difference

Example

A	B	C
a	b	c
d	a	f
c	b	d

relation R

D	E	F
b	g	a
d	a	f

relation S

A	B	C
a	b	c
c	b	d

R – S

Cartesian Product

- Cartesian product of two relations R and S, with degree k_1 and k_2, respectively, indicated

  R X S

  is a relation with degree $k_1 + k_2$ composed of all possible tuples such that:

  - their first k_1 components are tuples of R, and
  - their last k_2 components are tuples of S

- in the returned relation, the names of the first k_1 attributes are the names of attributes of relation R and the names of the last k_2 attributes are the names of the attributes of relation S

- if the two relations have attributes with same name <u>it is necessary to rename</u> those attributes in one of the two relations (more on renaming later)

Example

A	B	C
a	b	c
d	a	f
c	b	d

relation R

D	E	F
b	g	a
d	a	f

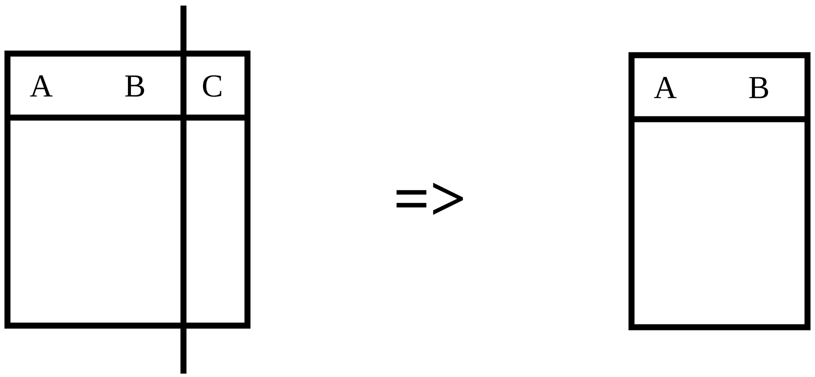
relation S

A	B	C	D	E	F
a	b	c	b	g	a
a	b	c	d	a	f
d	a	f	b	g	a
d	a	f	d	a	f
c	b	d	b	g	a
c	b	d	d	a	f

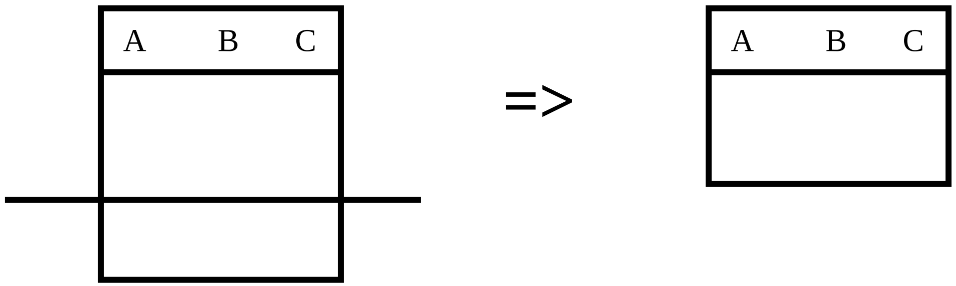
R X S

Projection and Selection

Projection = vertical decomposition



Selection = horizontal decomposition



Projection

- projection of a relation R on a set $A=\{A_1, A_2,....,A_m\}$ of attributes, indicated

$$\Pi_{A1, A2,....,Am}(R)$$

is a relation of degree *m* whose tuples have only attributes specified in A

- projection operation generates a set T of *m*-tuples (i.e., tuples with *m* attributes)

let $t = [A_1:v_1, A_2:v_2,....,A_m:v_m]$ be a *m*-tuple in T

t is such that there exists a tuple t' in R such that:

$$\forall A_i \in A \quad t[A_i] = t'[A_i]$$

- projection generates, from a given relation, a relation containing only a <u>subset of attributes</u>
- in the returned relation attributes are ordered according to the order specified in A

Example

A	B	C
a	b	c
d	a	f
c	b	d

Relation R

A	C
a	c
d	f
c	d

$\Pi_{A,C}(R)$

B	A
b	a
a	d
b	c

$\Pi_{B,A}(R)$

Selection: predicates

- a predicate F on a relation can be one of the following:
  - simple predicate
  - Boolean combination of simple predicates by means of logical connectives

    $\wedge$ (AND), $\vee$ (OR), $\neg$ (NOT)

- a *simple predicate* can be
  (i) A *op constant*
  (ii) A *op* A'

  where A and A' are attributes of R;
  op is a comparison operator: <, >, $\leq$, $\geq$, =, etc.
  constant is a constant value compatible with the domain of A

- examples:

B=b	simple predicate (i)
A=C	simple predicate (ii)
B=b $\vee$ A=C	Boolean combination
B=b $\wedge$ A=C	Boolean combination
$\neg$B=b	Boolean combination

Selection

- Selection on a relation R, given a predicate F, indicated σ_F (R)

  is a relation that contains all tuples satisfying predicate F
- the degree of the returned relation is the same as the degree of the original relation; the names of its attributes are the same as the name of the original relation
- if no tuple of R satisfies F, the result is an empty relation (indicated 0 or Ø)
- if k is the degree of R, selection generates a set T of k-tuples

  let $t = [A_1:v_1, A_2:v_2,....,A_k:v_k]$ be a k-tuple in T
  t is such that:

  $F\ (A_1/t[A_1], A_2/t[A_2],.....,A_k/t[A_k])$ is true,

where $A_i/t[A_i]$, $i=1,..,k$

denotes the substitution in F of the name of attribute A_i (if such name appears in F) with the value of the attributes named A_i in t

Example

relation R

A	B	C
a	b	c
d	a	f
c	b	d

A	B	C
a	b	c
c	b	d

$\sigma_{B=b}(R)$

A	B	C
d	a	f

$\sigma_{\neg(B=b)}(R)$

A	B	C
a	b	c
c	b	d

$\sigma_{B=b \vee A=C}(R)$

$\sigma_{B=b \wedge A=C}(R) = \emptyset$

Example

Employees

Emp#	Name	Job	Start_Date	Salary	Bonus	Dept#
7369	Rossi	engineer	17-Dec-90	1600,00	500,00	20
7499	Andrei	technician	20-Feb-91	800,00	?	30
7521	Bianchi	technician	20-Feb-91	800,00	100,00	30
7566	Rosi	manager	02-Apr-91	2975,00	?	20
7654	Martini	secretary	28-Sep-91	800,00	?	30
7698	Blacchi	manager	01-May-91	2850,00	?	30
7782	Neri	engineer	01-Jun-91	2450,00	200,00	10
7788	Scotti	secretary	09-Nov-91	800,00	?	20
7839	Dare	engineer	17-Nov-91	2600,00	300,00	10
7844	Turni	technician	08-Sep-91	1500,00	?	30
7876	Adami	engineer	28-Sep-91	1100,00	500,00	20
7900	Gianni	engineer	03-Dec-91	1950,00	?	30
7902	Fordi	secretary	03-Dec-91	1000,00	?	20
7934	Milli	engineer	23-Jan-92	1300,00	150,00	10
7977	Verdi	manager	10-Dec-90	3000,00	?	10

Departments

Dept#	Name_Dept	Office	Division	Manager
10	Civil Engineering	1100	D1	7977
20	R&D	2200	D1	7566
30	Surveying	5100	D2	7698

EXAMPLES

- **Q1: find the name of employees that have salary greater than 2000**

$$\Pi_{Name}(\sigma_{Salary>2000}(Employees))$$

Name
Rosi
Blacchi
Neri
Dare
Verdi

- **Q2: find the name and numbers of department of employees that are engineers and have salary greater than 2000**

$$\Pi_{Name,Dept\#}(\sigma_{Salary>2000 \wedge Job='engineer'}(Employees))$$

Name	Dep#
Neri	10
Dare	10

- **Q3: find the employee number of employees that:** (a) work in department 30 and (b) are engineers or technicians

$$\Pi_{Emp\#}(\sigma_{Dept\#=30 \wedge (Job='engineer' \vee Job='technician')}(Employees))$$

Emp#
7499
7521
7844
7900

Renaming

Renaming of a relation R with respect to a list of pairs of names of attributes

$$(A_1, B_1), (A_2, B_2), \ldots\ldots, (A_m, B_m)$$

such that A_i (i=1,...,m) is a name of an attribute in R, is denoted

$$\rho_{A1,\ A2,\ \ldots,\ Am\ \leftarrow B1,\ B2,\ \ldots,\ Bm}(R)$$

and renames attribute named A_i (i=1,...,m) with name B_i

Renaming is correct if the attributes of the new schema of relation R all have distinct names

Example:

R(A,B,C)

$\rho_{A,\ B,\ C\ \leftarrow AA,\ BB,\ CC}(R)$

modifies the schema of relation R to R(AA,BB,CC)

Basic Operations: Semantics

Let R = (A1, ..., Ak) be a relation schema, where Ai is a name of an attribute with domain Si, with i = 1 ...k.

We indicate $\Re(R)$ the set of all relations on that schema

- $\_ \cup \_ : \ \Re(R) \times \Re(R) \rightarrow \Re(R)$
  $r1 \cup r2 = \{t \mid t \in r1 \vee t \in r2\}$

- $\_ - \_ : \ \Re(R) \times \Re(R) \rightarrow \Re(R)$
  $r1 - r2 = \{t \mid t \in r1, t \notin r2\}$

- $\_ \text{ X } \_ : \ \Re(R1) \times \Re(R2) \rightarrow \Re(R1 \cdot R2)$
  $r1 \text{ X } r2 = \{t1 \cdot t2 \mid t1 \in r1, t2 \in r2\}$

- $\pi_{R'}\ \_ : \Re(R) \rightarrow \Re(R')$ with $R \supset R'$

  $\pi_{R'}(r) = \{t[R'] \mid t \in r\}$

- $\sigma_F\ \_ : \Re(R) \rightarrow \Re(R)$

  $\sigma_F(r) = \{t \mid t \in r, F(t)\}$

Derived operations : Join

- join of two relations R and S on attributes A of R and A' of S, indicated

$$R \underset{A\theta A'}{\bowtie} S$$

  is defined as $\sigma_{A\theta A'}(R \times S)$

- join is a Cartesian product followed by a selection; $A\theta A'$ is called *join predicate*

- the degree of the resulting relation is the sum of the degrees of the original relations

Examples

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

relation R

D	E
3	1
6	2

relation S

A	B	C	D	E
1	2	3	3	1

$R \underset{A=E}{\bowtie} S$

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

$R \underset{B<D}{\bowtie} S$

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

R X S

A	B	C	D	E
1	2	3	3	1

$\sigma_{A=E}$ (R X S)

$R \underset{A=E}{\bowtie} S$

Natural Join

- Natural join is a particular case of join

- Example: “find the name of all employees and the office in which they work”

  We can express this query by joining Employees and Departments based on the predicate:

  Employees.Dept# = Departments.Dept#

- this particular case of join is based on the equality of all attributes common to the two relations

- joins based on equality of attributes are very frequently used

- in this case we can omit the predicate

Natural Join

We can express the previous query as: $\Pi_{\text{Name, Office}}(\text{Employees} \bowtie \text{Departments})$

Definition

- let R and S be relations
- let {A1,A2,....,A*k*} = $U_R \cap U_S$ be the set of attributes common both to the schema of R and the schema of S
- let {I1,I2,....,I*m*} = $U_R \cup U_S$ be the union of attributes in the schema of R and in the schema of S

the expression that defines natural join is

$$R \bowtie S = \Pi_{I1,I2,....,Im}(\sigma_C (R \times (\rho_{A1,\, A2,\, ...,Ak \leftarrow S.A1,\, S.A2,...\, S.Ak}(S))))$$

where C is a predicate $A1{=}S.A1 \ \textbf{AND} \ A2{=}S.A2 \ \textbf{AND} \ .... \ Ak{=}S.Ak$

- natural join performs a join based on the equality of attributes common to the two relations and then eliminates all duplicate attributes ie. in our example only one of the columns Dept# appears in the result (and there is no need to use Renaming)

Example

A	B	C
a	b	c
d	b	c
b	b	f
c	a	d

R

B	C	D
b	c	d
b	c	e
a	d	b

S

A	B	C	D
a	b	c	d
a	b	c	e
d	b	c	d
d	b	c	e
c	a	d	b

$R \bowtie S$

Derived Operations: Semantics

Let R = (A1, ..., A*k*) be a relation schema, with Ai name of attribute with domain S*i*, *i* = 1 ...*k*.
We indicate with $\Re$(R) the set of all relations on such schema

- $\_ \cap \_ : \Re(R) \times \Re(R) \rightarrow \Re(R)$

  $r1 \cap r2 = r1 - (r1 - r2) = \{t \mid t \in r1, t \in r2\}$

- $\_ \bowtie_F \_ : \Re(R1) \times \Re(R2) \rightarrow \Re(R1 \cdot R2)$

  $r1 \bowtie_F r2 = \sigma_F (r1 \ \ X \ \ r2) =$

  $\{t1 \cdot t2 \mid t1 \in r1, t2 \in r2, F(t1,t2)\}$

- $\_\bowtie\_ : \Re(R1) \times \Re(R2) \rightarrow \Re(R1 \cup R2)$

  $r1 \bowtie r2 =$

  $\{t \mid t[R1] \in r1, t[R2] \in r2\}$

  - if $R1 \cap R2 = \varnothing$ $\quad r1 \bowtie r2 = r1 \times r2$
  - if $R1 = R2$ $\quad r1 \bowtie r2 = r1 \cap r2$

Relational Calculus

- Relational Algebra is a "procedural" language: to specify an algebraic expression, we indicate operations that must be performed to generate the query result
- Relational Calculus: we provide a formal description of the result without specifying how to obtain it (“declarative” language)
- two alternatives:

  - tuple relational calculus (TRC) = variables represent tuples (we study this version)

  - domain relational calculus (DRC) = variables represent domains

Relational Calculus

In TRC a query is an expression:

$\{t: U \mid P(t)\}$

ie. It is defined as the set of tuples t on a set U of attributes such that t satisfies predicate P

Notation t[A] indicates the value of attribute A in t

(example: t[Name])

$t \in R$ indicates that tuple t is in relation R

Examples:

- **find all employees whose salary is greater than 2000**

$\{t: U_{Employees} \mid t \in Employees \land t[Salary] > 2000\}$

- **find the name of all employees whose salary is greater than 2000**

  $\{t: \{Name\} \mid (\exists s)\ (s \in Employees \wedge s[Salary] > 2000 \wedge s[Name] = t[Name])\}$

  t represents a variable that indicates tuples belonging to a relation with schema = {Name} notation $(\exists t)(Q(t))$ indicates that there exists a tuple t such that $Q(t)$ is true

- **find names and offices of employees whose salary is greater than 2000**

  $\{t: \{Name, Office\} \mid (\exists s)\ (s \in Employees \wedge s[Salary] > 2000 \wedge s[Name] = t[Name] \wedge$
  $(\exists u)\ (u \in Departments \wedge s[Dept\#] = u[Dept\#] \wedge u[Office] = t[Office]))\}$

- **find names of employees that either have a salary greater than 2000 or work in a department of division D1**

  $\{t: \{Name\} \mid (\exists s)\ (s \in Employees \wedge s[Name] = t[Name]$
  $\wedge (s[Salary] > 2000 \vee (\exists u)\ (u \in Departments \wedge s[Dept\#] = u[Dept\#] \wedge u[Division] = "D1")))\}$

Relational Calculus

Operations of relational algebra are expressed as:

- $R \cup S$ **Union**

  $\{t : U_R \mid t \in R \vee t \in S\}$

- $R - S$ **Difference**

  $\{t : U_R \mid t \in R \wedge t \notin S\}$

- R X S **Cartesian Product**

  with U_R ={A1, A2, ..., An}

  U_S ={A1', A2', ..., Am'}

$\{t: \{U_R \cup U_S\} \mid (\exists x)(\exists y)\ (x \in R \wedge y \in S \wedge$

$x[A1] = t[A1] \wedge x[A2] = t[A2] \wedge \ldots\ldots \wedge\ x[An] = t[An] \wedge$

$y[A1'] = t[A1'] \wedge y[A2'] = t[A2'] \wedge \ldots\ldots \wedge\ y[Am'] = t[Am'])\}$

Relational Calculus

- $\Pi_{A1,A2,\ ...Ak}(R)$ **Projection**

  $\{t : \{A1, A2,....,Ak\} \mid (\exists x)\ (x \in R \wedge$

  $x[A1] = t[A1] \wedge x[A2] = t[A2] \wedge ..... \wedge x[Ak] = t[Ak])\}$

- $\sigma_F\ (R)$ **Selection**

  $\{t: U_R \mid t \in R \wedge F'\}$

  where F' is formula F where each attribute A has been replaced by t[A]

Relational Calculus

Expressions in relational calculus are also called FORMULAS and they are of the form:

$\{t: U \mid P(T)\}$ set of tuples on a schema U that satisfy predicate P

In other words, an answer tuple is an assignment of constant values to variables that make the formula evaluate true

UNSAFE QUERIES AND EXPRESSIVE POWER

Possible to write syntactically correct calculus queries with infinite number of answers.

Such queries are called UNSAFE.

Example: $\{t: U_R \mid \neg (t \in R)\}$

However, it has been shown that every query that can be expressed in relational algebra can be expressed as a safe query in relational calculus (TRC/DRC); the converse is also true.

=> same expressive power.

Relational Completeness: Relational query languages (e.g., SQL) can express every query that can be expressed in relational algebra/calculus.

Semantic Integrity Constraints

A constraint is a property that a set of data must satisfy. One possible classification of constraints:

- *immediate*: verified immediately after each modification of the DB
- *deferred*: verified only at the end of a series of operations (transaction)

• constraints can also be classified depending on the objects they access:

- on a single relation
    - (i) on a single tuple:
        - * attribute constraints
        - * multiple attribute constraints
    - (ii) on multiple tuples of the same relation
        - * functional dependencies
        - * cardinality constraints
    - (iii) aggregation constraints
- on multiple relations: referential integrity

Examples:

- on a single attribute:

  salary of an employee must be between 500 and 1000

- on multiple attributes:

  bonus of an employee must always be less than the salary

- cardinality constraints:

  there must be at least 3 technicians (i.e., 3 employees whose job = “technician”)

- aggregation constraints:

  the average salary for a technician must be greater than 500

- constraints on multiple relations:

  the sum of salaries of employees that work on project P must be less than the budget for P