The Relational Data Model

The **relational model** (E. Codd, 1970) supports powerful, simple and declarative languages with which operations on data are expressed. We define operations on relations whose results are themselves relations.

Domain

A **domain** is a set of values.

Examples:

- a set of integers is a domain
- set of character strings
- set of character strings of length 20
- set of real numbers
- set {0,1}

Cartesian Product

The **Cartesian product** (or just **product)** of domains $D_1,D_2,\ldots,D_k$, written $D_1 x D_2 x \ldots x D_k$, is the set of all k-tuples $(d_1,d_2,\ldots,d_k)$ such that d_1 is in D_1, d_2 is in D_2, and so on.

Example:

$k=2$ $D_1=\{0,1\}$ $D_2=\{a,b,c\}$

$D_1 x D_2 = \{(0,a),(0,b),(0,c),(1,a),(1,b),(1,c)\}$

Relation

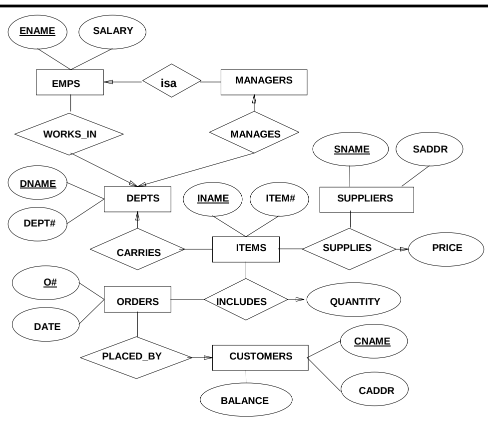
A **relation** is any subset of the Cartesian product of one or more domains. We shall assume that a relation is finite unless we state otherwise.

Example:

$\{(0,a),(0,c),(1,b)\}$ – subset of $D_1 x D_2$

The members of a relation are called **tuples**. Each relation that is a subset of some product $D_1 x D_2 x \ldots x D_k$ of k domains is said to have **arity** (or **degree**) k. A tuple $(d_1,d_2,\ldots,d_k)$ has k components: the ith component is d_i. A tuple with k components is called a **k-tuple**.
It helps to view a relation as a table, where each row is a tuple and each column corresponds to one component. The columns are often given names, called **attributes**. The set of attributes names for a relation is called the **relation scheme**. If we name a relation REL, and its relation scheme has attributes $A_1,A_2,\ldots,A_k$, we often write the relation scheme as REL $(A_1,A_2,\ldots,A_k)$.

The collection of relation schemes used to represent information is called a (**relational**) **database scheme**, and the current values of the corresponding relations form the (**relational**) **database**.



The data of an entity-relationship diagram is represented by two sorts of relations:

1. An entity set E can be represented by a relation whose relation scheme consists of all the attributes of the entity set. Each tuple of the relation represents one entity in the current instance of E.
   Example: The entity set CUSTOMERS is represented by the relation
   CUSTOMERS (CNAME, CADDR, BALANCE)

If E is an entity set whose entities are identified through a relationship with some other entity set F, then the relational scheme also has the attributes of F that are needed for the key of E.

Example: **The relation for entity set** MANAGERS **has only one attribute,** ENAME**, which is the key for** MANAGERS**. The value of** ENAME **for a given manager is the name of the employee entity that is this manager.**

MANAGERS (ENAME)

2. A relationship R among entities $E_1, E_2, \ldots, E_k$ is represented by a relation whose relational scheme consists of the attributes in the keys for each of $E_1, E_2, \ldots, E_k$. By renaming attributes if necessary, we make certain that no two entity sets in the list have attributes with the same name, even if they are the same entity set.

The relation scheme for the entity sets is (each entity set has the same name as the relation):

1. EMPS (ENAME, SALARY)
2. MANAGERS (ENAME)
3. DEPTS (DNAME, DEPT#)
4. SUPPLIERS (SNAME, SADDR)
5. ITEMS (INAME, ITEM#)
6. ORDERS (O#, DATE)
7. CUSTOMERS (CNAME, CADDR, BALANCE)

Now let us consider the relationships. We should not create a relation for the isa **relationship, since it would just consist of the** ENAME **attribute repeated (and renamed in one repetition). And would hold exactly the same information as the** MANAGES **relation; that is, it would list the names of all those employees who are managers. The other six relationship yield the following relation schemes:**

8. WORKS_IN (ENAME, DNAME)
9. MANAGES (ENAME, DNAME)
10. CARRIES (INAME, DNAME)
11. SUPPLIES (SNAME, INAME, PRICE)
12. INCLUDES (O#, INAME, QUANTITY)
13. PLACED_BY (O#, CNAME)

In each case, the set of attributes is the set of keys for the entity sets connected by the relationship of the same name as the relation.

Example: SUPPLIES **connects** SUPPLIERS**,** ITEMS**, and** PRICE**, which have keys** SNAME**,** INAME**, and** PRICE**, respectively, and it is these three attributes we see in the scheme for YVCB.**

SUPPLIES (SNAME, INAME, PRICE)

The two relations MANAGES **and** WORKS_IN **have the same set of attributes, but of course their meaning are different. The tuple** (e, d) **in** MANAGES **means that** e **manages department** d**, while the same tuple in** WORKS_IN **means that** e **is an employee in department** d**.**

Keys of Relations

A set S of attributes of a relation R is a key **if**

1. No instance of R that represents a possible state of the world can have two tuples that agree in all the attributes of S, yet are not the same tuple, and

2. No proper subset of S has property (1).

Example:

In the relation SUPPLIES**,** SNAME **and** INAME **together form a key. If there are two tuples** (s,i,p_1) **and** (s,i,p_2) **in** SUPPLIES**, then supplier** s **would apparently sell item** i **both at price** p_1 **and at price** p_2**, a situation that means our data is faulty. This observation justifies condition (1).**

To check (2) we have to consider the proper subset, that is SNAME alone and INAME alone. Neither should satisfy condition (1). For example, it is quite possible that we find the two tuples

(Acme, Brie, 3.50)
(Acme, Perrier, 1.25)

in SUPPLIES **at the same time, and although they agree on** SNAME**, they are not the same tuple. Similarly, we might find**

(Acme, Brie, 3.50)
(Ajax, Brie, 3.95)

showing that INAME **alone does not satisfy condition (1).**

A relation may have more than one key.
Example: Consider DEPTS (DNAME, DEPT#)**. We do not give two departments the same name, and we do not give two departments the same number, so we may declare that** DNAME **is a key and** DEPT# **is a different key. But it is useful to select one unique key.**

Primary key is a unique key selected from among several choices, all of which are called candidate keys.

The rules are:

1. **If a relation comes from an entity set, a set of attributes is a key for that relation if it is a key for the entity set.**
2. **If a relation comes from a many-many relationship, then the key for the relation is normally the set of all the attributes.**
3. **If a relation comes from a one-to-one relationship between entity set** E **and** F**, than the key for** E **and the key for** F **are both keys for the relation. That relations, like entity sets, can have more that one set of attributes that is a candidate key.**
4. **If a relation comes from a relationship that is many-one from** $E_1, E_2, \ldots, E_{k-1}$ **to** E_k**, then the set of attributes that is the union of the keys for** $E_1, E_2, \ldots, E_{k-1}$ **is normally a key for the relation.**

Examples:
In SUPPLIES **the lone key consists of two attributes, since the relationship is many-one from** SUPPLIERS **and** ITEMS **to** PRICE **(4), and the first two entity sets have keys** SNAME **and** INAME**, respectively. Thus,** {SNAME, INAME} **forms a key for relation** SUPPLIES**.**

The relation DEPTS **has two candidate keys, each consisting of one attribute.** DNAME **is a key for entity set** DEPTS**, but we might well decide that** DEPT# **also should be a key, since the YVCB probably does not intend to give two departments the same number.**

Relations with Common Keys

When two relations have a candidate key in common, we can combine the attributes and receive a relation whose set of attributes is the union of the two sets.

Example:
Relations DEPTS and MANAGES each have DNAME as a candidate key; in once case it is the primary key and in the other not. We may thus replace DEPTS and MANAGES by one relation

DEPTS (DNAME, DEPT#, MGR)

The new relation has the same name DEPTS**. The attributes** DNAME **and** DEPT# **are the same as the attributes of the same name in the old** DEPTS **relation, while** MGR **is intended to be the attribute** ENAME **from** MANAGES**.**

Dangling Tuples

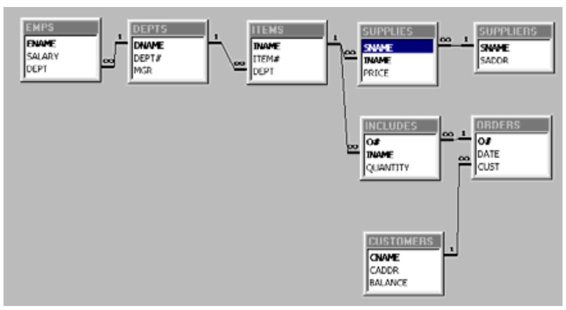
Tuples that need to share a value with a tuple in another relation, but find no such value, are called dangling tuples. To avoid this problem we add to the database scheme information about existence constraints: if a tuple v **appears in attribute** A **of some tuple in relation** R**, then** v **must also appear in attribute** B **of same tuple in relation** S**. We can store null values in certain fields. This null value may appear if the field is not a primary key.**
Example:
If the YVCB has a Wine department, whose number is 16, but that temporarily has no manager, we will represent this data with null value for the MGR **attribute. If we add a manager Truffle of the Gourmet department, whose number is not yet assigned, we could represent this with null value for the** DEPT# **attribute.**

Example:

The new combined simplified list of relations:

1.	EMPS (<u>ENAME</u>, SALARY, DEPT)	**1,8**
2.	DEPTS (<u>DNAME</u>, DEPT#, MGR)	**2,3,9**
3.	SUPPLIERS (<u>SNAME</u>, SADDR)	**4**
4.	ITEMS (<u>INAME</u>, ITEM#, DEPT)	**5,10**
5.	ORDERS (<u>O#</u>, DATE, CUST)	**6,13**
6.	CUSTOMERS (<u>CNAME</u>, CADDR, BALANCE)	**7**
7.	SUPPLIES (<u>SNAME</u>, <u>INAME</u>, PRICE)	**11**
8.	INCLUDES (<u>O#</u>, <u>INAME</u>, QUANTITY)	**12**

We can see that the new relation DEPTS combines MANAGERS, DEPTS, and MANAGES. DEPTS and MANAGES shared the common candidate key DNAME. However, MANAGERS, with key, ENAME, does not share a common candidate key with these. But MANAGES is a one-to-one relationship between ENAME and DNAME. Hence, these two attributes are in a sense equivalent, we may regard MANAGERS as if its attributes were DNAME rather than ENAME.



Relationships

Operations in the Relational Data Model

A relation is a set of k-tuples for some k, called the arity of the relation. The operands of relational algebra are either constant relations or variables denoting relations of a fixed arity.

1. Union. The union of relations R and S, denoted RUS, is the set of tuples that are in R or S or both. R and S are relations of the same arity.

Example:

A B C	D E F	a b c
a b c	b g a	d a f
d a f	d a f	c b d
c b d		b g a
R	**S**	**RUS**

2. Set difference. The difference of relations R and S, denoted R-S, is the set of tuples in R but not in S. R and S have the same arity.

Example:

A B C	D E F	a b c
a b c	b g a	c b d
d a f	d a f	
c b d		
R	**S**	**R-S**

3. Cartesian product. Let R and S be relations of arity k_1 and k_2, respectively. Then RxS, the product of R and S, is the set of all possible (k_1+k_2)-tuples whose first k_1 components form a tuple in R and whose last k_2 components form a tuple in S.

Example:

A B C	D E F	A B C D E F
a b c	b g a	a b c b g a
d a f	d a f	a b c d a f
c b d		d a f b g a
R	**S**	d a f d a f
		c b d b g a
		c b d d a f
		RxS

4. Projection. We take a relation R, remove some of the components (attributes) and/or rearrange some of the remaining components. If R is a relation of arity k, we let $\Pi_{i_1,i_2,\ldots,i_m}(R)$, where the i_j's are distinct integers in the range 1 to k. If R has attributes labeling its columns, then we may use the same attribute names in the projected relation.

Example:

R

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

$\Pi_{C,A}(R)$ or $\Pi_{3,1}(R)$

C	A
c	a
f	d
d	c

5. Selection. Let F be a formula involving

a) Operands that are constants or component numbers;

b) The arithmetic comparison operators <, =, >, ≤, ≠, and ≥, and

c) The logical operations ∧ (and), ∨ (or) and ¬ (not).

Then $\sigma_F(R)$ is the set of tuples in R such that when, for all i, we substitute the ith component of for any occurrences of \$i in formula F, the formula F becomes true.

Example:

R

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

$\sigma_{\$2=b}(R)$ or $\sigma_{B=b}(R)$

A	B	C
a	b	c
c	b	d

6. Intersection. Intersection R∩S is the set of tuples that are both in R and S; it is equivalent to R-(R-S).

Example:

R

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

S

D	E	F
b	g	a
d	a	f

R∩S

d	a	f

7. Join. The Θ-join of R and S on columns i and j, written $R \underset{i\Theta j}{\infty} S$, where Θ is an arithmetic comparison operator (=, <, and so on), is shorthand for $\sigma_{\$i\Theta\$(r+j)}(RxS)$, if R is of arity r. That is, the Θ-join of R and S is those tuples in the product of R and S such that the ith component of R stands in relation to the jth component of S. If Θ is =, the operation is often call an **equijoin**.

Example:

R

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

S

D	E
3	1
6	2

RxS

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

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

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

8. Natural join. The natural join R∞S is applicable only when both R and S have columns that are named by attributes.

a) Compute RxS.

b) For each attribute A that names both a column in R and a column in S select from RxS those tuples whose values agree in the columns for R.A and S.A.

c) For each attribute A above, project out the column S.A, and call the remaining column, R.A, simply A.

Example:

R

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

S

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

RxS

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

R∞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

9. Semijoin. The semijoin of relation R by relation S, written R∝S, is the projection onto the attributes of R of the natural join of R and S.

Example:

R

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

S

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

R∞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∝S

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

Relational Algebra as a Query Language

Example:

Consider the relation SUPPLIES. Which suppliers supply Brie?

$\Pi_{SNAME}(\sigma_{INAME=\text{“Brie”}}(SUPPLIES))$

The result will be a list of all suppliers of Brie.

Example:

What items supplier ‘Acme’ sells for less than $5, and the prices of each?

$\Pi_{INAME,PRICE}(\sigma_{SNAME=\text{“Acme”} \wedge PRICE<5}(SUPPLIES))$