DataBase Chapter Two Outline

The set of allowed values for each attribute is called the **domain** of the attribute

*r*(*R*) is a *relation* on the *relation schema R*

Example:

*customer (Customer\_schema)*

About Key:

*K* is a **superkey** of *R* if values for *K* are sufficient to identify a unique tuple of each possible relation *r(R)*

*K* is a **candidate key** if *K* is minimal (最小超码)

**Primary Key :** 被数据库设计者选中的用来在同一关系中区分不同元组的**候选码**。

**Relational Algebra**

Procedural language

Six basic operators

select: σ

project: ∏

union: ∪

set difference: *–*

Cartesian product: x

rename: *ρ*

The operators take one or two relations as inputs and produce a new relation as a result.

**Select Operation**

Notation: *σ* *p*(*r*)

*p* is called the **selection predicate**

Defined as:  
 *σp*(***r***) = {*t* | *t* ∈ *r* **and** *p(t)*}

Where *p* is a formula in propositional calculus consisting of **terms** connected by : ∧ (**and**), ∨ (**or**), ¬ (**not**)  
Each **term** is one of:

<attribute> *op* <attribute> or <constant>

where *op* is one of: =, ≠, >, ≥. <. ≤

**Project Operation**

Notation:

where *A1, A2* are attribute names and *r* is a relation name.

**Union Operation**

Notation: *r* ∪ *s*

Defined as:

*r* ∪ *s* = {*t* | *t* ∈ *r* or *t* ∈ *s*}

For *r* ∪ *s* to be valid.

1. *r,* *s* must have the *same* **arity** (same number of attributes)

2. The attribute domains must be **compatible** (example: 2nd column of *r* deals with the same type of values as does the 2nd column of *s*)

**Set Difference Operation**

Notation *r – s*

Defined as:

*r – s* = {*t* | *t* ∈ *r* **and** t ∉ *s*}

Set differences must be taken between **compatible** relations.

*r* and *s* must have the same arity

attribute domains of *r* and *s* must be compatible

**Cartesian-Product Operation**

Notation *r* x *s*

Defined as:

*r* x *s* = {*t q* | *t* ∈ *r* **and** *q* ∈ *s*}

Assume that attributes of r(R) and s(S) are disjoint. (That is, *R* ∩ *S* = *∅*).

If attributes of *r(R)* and *s(S*) are not disjoint, then renaming must be used.

**Rename Operation**

*Allows us to name, and therefore to refer to, the results of relational-algebra expressions.*

*Allows us to refer to a relation by more than one name.*

*Example:*

*ρ x (E)*

*returns the expression E under the name X*

*If a relational-algebra expression E has arity n, then*



*returns the result of expression E under the name X, and with the*

*attributes renamed to A1 , A2 , …., An .*

**Additional Operations**

**Set-Intersection Operation**

Notation: *r* ∩ *s*

Defined as:

*r* ∩ *s* = { *t* | *t* ∈ *r* **and** *t* ∈ *s* }

Assume:

*r*, *s* have the *same arity*

attributes of *r* and *s* are compatible

Note: *r* ∩ *s* = *r* – (*r* – *s*)

**Natural-Join Operation**

Notation: rs

*Let r and s be relations on schemas R and S respectively.   
Then, r s is a relation on schema R ∪ S obtained as follows:*

*Consider each pair of tuples tr from r and ts from s.*

*If tr and ts have the same value on each of the attributes in R ∩ S, add a tuple t to the result, where*

*t has the same value as tr on r*

*t has the same value as ts on s*

**θ -Join Operation**

rθs= σ θ(*r* x *s*)

**Division Operation**

Notation:

Suited to queries that include the phrase “for all”.

Let *r* and *s* be relations on schemas *R* and *S* respectively where

*R* = (*A*1, …, *Am* , *B*1, …, *Bn* )

*S* = (*B*1, …, *Bn*)

The result of r ÷ s is a relation on schema

*R* – *S* = (*A*1, …, *Am*)

*r* ÷ *s* = { *t* | *t* ∈ ∏ *R-S* (*r*) ∧ ∀ *u* ∈ *s* ( *tu* ∈ *r* ) }

Where *tu* means the concatenation of tuples *t* and *u* to produce a single tuple

*r* ÷ *s* = ∏*R-S* (*r* ) – ∏*R-S* ( ( ∏*R-S* (*r* ) x *s* ) – ∏*R-S,S*(*r* ))

**Assignment Operation**

The assignment operation (←) provides a convenient way to express complex queries.

Write query as a sequential program consisting of

* + - a series of assignments
    - followed by an expression whose value is displayed as a result of the query.

Assignment must always be made to a temporary relation variable.

Example: Write *r* ÷ *s* as

*temp1*← ∏*R-S* (*r* )   
 *temp2* ← ∏*R-S* ((*temp1* x *s* ) – ∏*R-S,S* (*r* ))  
 *result* ← *temp1* – *temp2*

The result to the right of the ← is assigned to the relation variable on the left of the ←.

May use variable in subsequent expressions.

**Extended Relational-Algebra-Operations**

**Generalized Projection** 允许投影使用算数运算跟重命名。

Extends the projection operation by allowing arithmetic functions to be used in the projection list.  


*E* is any relational-algebra expression

Each of *F*1, *F*2, …, *Fn* are are arithmetic expressions involving constants and attributes in the schema of *E*.

Given relation *credit\_info(customer\_name, limit, credit\_balance),* find how much more each person can spend:

∏*customer\_name, limit – credit\_balance (credit\_info)*

*Also, we can apply rename operation:*

∏*customer\_name, limit – credit\_balance* ***as*** *credit\_availble (credit\_info)*

**Aggregate Functions and Operations** 允许使用列函数，并且分组。

**Aggregation function** takes a collection of values and returns a single value as a result.

**avg**: average value  
 **min**: minimum value  
 **max**: maximum value  
 **sum**: sum of values  
 **count**: number of values

**Aggregate operation** in relational algebra



*E* is any relational-algebra expression

*G1*, *G2* …, *Gn* is a list of attributes on which to group (can be empty)

Each *Fi* is an aggregate function

Each *Ai* is an attribute name

The schema of result is (*G1*, *G2* …, *Gn* , *a1*, *a2* …, *am* ).where a*i* is the result of applying the aggregate function F*i* on the multiset of values for attribute A*i* in the group

**Outer Join**

An extension of the join operation that avoids loss of information.

Computes the join and then adds tuples form one relation that does not match tuples in the other relation to the result of the join.

Uses *null* values:

*null* signifies that the value is unknown or does not exist

All comparisons involving *null* are (roughly speaking) **false** by definition.

* + - We shall study precise meaning of comparisons with nulls later

**The result of any arithmetic expression involving *null* is *null.***

**Modification of the Database**

**Deletion**

**Can delete only whole tuples; cannot delete values on only particular attributes**

**A deletion is expressed in relational algebra by:**

***r* ← *r* – *E***

**where *r* is a relation and *E* is a relational algebra query.**

**Insertion**

**To insert data into a relation, we either:**

**specify a tuple to be inserted**

**write a query whose result is a set of tuples to be inserted**

**in relational algebra, an insertion is expressed by:**

***r* ← *r* ∪ *E***

**where *r* is a relation and *E* is a relational algebra expression.**

**The insertion of a single tuple is expressed by letting *E* be a constant relation containing one tuple.**

**Updating**

**A mechanism to change a value in a tuple without charging *all* values in the tuple**

**Use the generalized projection operator to do this task**



**Each *Fi* is either**

**the *I* th attribute of *r*, if the *I* th attribute is not updated, or,**

**if the attribute is to be updated F*i* is an expression, involving only constants and the attributes of *r*, which gives the new value for the attribute**