Relational Databases

- Basic concepts
  - Data model: organize data as tables
  - A relational database is a set of tables
- Advantages
  - Simple concepts
  - Solid mathematical foundation
    - set theory
  - Powerful query languages
  - Efficient query optimization strategies
  - Design theory
- Industry standard
  - Relational model
  - SQL language

Relational Model

- Relation
  - A relation R with attributes $A=\{A_1, A_2, \ldots, A_n\}$ defined over n domains $D=\{D_1, D_2, ..., D_n\}$ (not necessarily distinct) with values $\{Dom_1, Dom_2, ..., Dom_n\}$ is a finite, time varying set of n-tuples $<d_1, d_2, ..., d_n>$ such that $d_1 \in Dom_1$, $d_2 \in Dom_2$, ..., $d_n \in Dom_n$ and $A_1 \in D_1$, $A_2 \in D_2$, ..., $A_n \in D_n$.
  - Notation: $R(A_1, A_2, \ldots, A_n)$ or $R(A_1: D_1, A_2: D_2, \ldots, A_n: D_n)$
  - Alternatively, given R as defined above, an instance of it at a given time is a set of n-tuples:

    $\{< A_1: d_1, A_2: d_2, \ldots, A_n: d_n> \mid d_1 \in Dom_1, d_2 \in Dom_2, ..., d_n \in Dom_n\}$
- Tabular structure of data where
  - R is the table heading
  - attributes are table columns
  - each tuple is a row

Relation Schemes and Instances

- Relational scheme
  - A relation scheme is the definition; i.e., a set of attributes
  - A relational database scheme is a set of relation schemes:
    - i.e., a set of sets of attributes
- Relation instance (simply *relation*)
  - An relation is an instance of a relation scheme
  - a relation **r** over a relation scheme $R = \{A_1, ..., A_n\}$ is a subset of the Cartesian product of the domains of all attributes, i.e.,

$$\mathbf{r} \subseteq Dom_1 \times Dom_2 \times \ldots \times Dom_n$$

Domains

- A domain is a *type* in the programming language sense
  - Name: String
  - Salary: Real
- Domain values is a set of acceptable values for a variable of a given type.
  - Name: CdnNames = {…},
  - Salary: ProfSalary = {45,000 - 150,000}
  - Simple/Composite domains
    - Address = Street name+street number+city+province+ postal code
- Domain compatibility
  - Binary operations (e.g., comparison to one another, addition, etc) can be performed on them.
- Full support for domains is not provided in many current relational DBMSs

Relation Schemes

EMP(ENO, ENAME, TITLE)

PROJ (PNO, PNAME, BUDGET)

WORKS(ENO,PNO, RESP, DUR)

PAY(TITLE, SALARY)

- Underlined attributes are relation keys (tuple identifiers).
- Tabular form

EMP

ENO	ENAME	TITLE

PROJ

PNO	PNAME	BUDGET

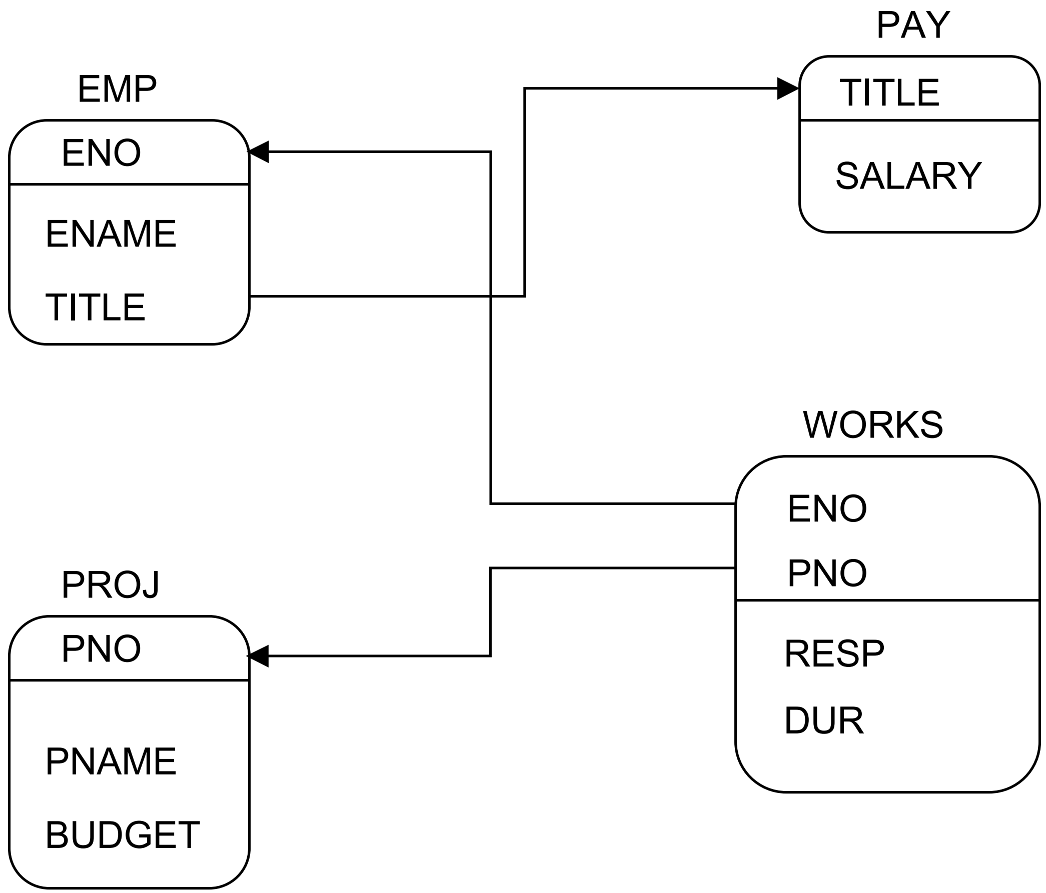
WORKS

ENO	PNO	RESP	DUR

PAY

TITLE	SALARY

Different Representation



Example Relation Instances

EMP

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng.
E2	M. Smith	Syst. Anal.
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E5	B. Casey	Syst. Anal.
E6	L. Chu	Elect. Eng.
E7	R. Davis	Mech. Eng.
E8	J. Jones	Syst. Anal.

WORKS

ENO	PNO	RESP	DUR
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P3	Consultant	10
E3	P4	Engineer	48
E4	P2	Programmer	18
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E7	P5	Engineer	23
E8	P3	Manager	40

PROJ

PNO	PNAME	BUDGET
P1	Instrumentation	150000
P2	Database Develop.	135000
P3	CAD/CAM	250000
P4	Maintenance	310000
P5	CAD/CAM	500000

PAY

TITLE	SALARY
Elect. Eng.	55000
Syst. Anal.	70000
Mech. Eng.	45000
Programmer	60000

Properties

- Based on finite set theory
  - No ordering among attributes
    - Sometimes we prefer to refer to them by their relative order
  - No ordering among tuples
    - Query results may be ordered, but two differently ordered relation instances are equivalent
  - No duplicate tuples allowed
    - Commercial systems allow duplicates (so bag semantics)
  - Value-oriented: tuples are identified by the attributes values
- All attribute values are atomic
  - no tuples, or sets, or other structures
- Degree or arity
  - number of attributes
- Cardinality
  - number of tuples

Integrity Constraints

- Key Constraints
  - Key: a set of attributes that uniquely identifies tuples
  - Candidate key: a minimum set of attributes that form a key
  - Superkey: A set of one or more attributes, which, taken collectively, allow us to identify uniquely a tuple in a relation.
  - Primary key: a designated candidate key
- Data Constraints
  - Functional dependency, multivalued dependency, …
  - Check constraints
- Others
  - Null constraints
  - Referential constraints

Views

- Views can be defined
  - on single relations PROJECT(PNO, PNAME)
  - on multiple relations SAL(ENO,TITLE,SALARY)
- Relations from which they are derived are called base relations
- View relations can be
  - virtual; never physically created
    - updates to views is a problem
  - materialized: physical relations exist
    - propagation of base table updates to materialized view tables

View Updates

- Views that are derived from multiple tables may cause problems

EMP

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng.
E2	M. Smith	Syst. Anal.
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E5	B. Casey	Syst. Anal.
E6	L. Chu	Elect. Eng.
E7	R. Davis	Mech. Eng.
E8	J. Jones	Syst. Anal.

PAY

TITLE	SALARY
Elect. Eng.	55000
Syst. Anal.	70000
Mech. Eng.	45000
Programmer	60000

⇨

SAL

ENO	TITLE	SALARY
E1	Elect. Eng.	55000
E2	Syst. Anal.	70000
E3	Mech. Eng.	45000
E4	Programmer	60000
E5	Syst. Anal.	70000
E6	Elect. Eng.	55000
E7	Mech. Eng.	45000
E8	Syst. Anal.	70000

How do you delete a tuple from SAL?

Relational Algebra

Form

$$<\mathit{Operator}>_{<\mathit{parameters}>} <\mathit{Operands}> \;\rightarrow\; <\mathit{Result}>$$

$\downarrow$ Operands: Relation (s)

$\downarrow$ Result: Relation

Relational Algebra Operators

- Fundamental
  - union
  - set difference
  - selection
  - projection
  - Cartesian product
- Additional
  - rename
  - intersection
  - join
  - quotient (division)
- Union compatibility
  - same degree
  - corresponding attributes defined over the same domain

Union

- Similar to set union
- General form

$$R \cup S = \{t \mid t \in R \text{ or } t \in S\}$$

where R, S are relations, t is a tuple variable

- Result contains tuples that are in R or in S, but not both (duplicates removed)
- R, S should be union-compatible

Set Difference

- General Form

$$R - S = \{t \mid t \in R \text{ and } t \notin S\}$$

where R and S are relations, t is a tuple variable

- Result contains all tuples that are in R, but not in S.
- $R - S \neq S - R$
- R, S union-compatible

Selection

- Produces a horizontal subset of the operand relation
- General form

$$\sigma_F(R)=\{t \mid t \in R \text{ and } F(t) \text{ is true}\}$$

where

- *R* is a relation, *t* is a tuple variable
- *F* is a formula consisting of
  - operands that are constants or attributes
  - arithmetic comparison operators

    $<, >, =, \neq, \leq, \geq$
  - logical operators

    $\wedge, \vee, \neg$

Selection Example

EMP

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng.
E2	M. Smith	Syst. Anal.
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E5	B. Casey	Syst. Anal.
E6	L. Chu	Elect. Eng.
E7	R. Davis	Mech. Eng.
E8	J. Jones	Syst. Anal.

$\sigma_{\text{TITLE='Elect. Eng.'}}(\text{EMP})$

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng
E6	L. Chu	Elect. Eng.

Projection

- Produces a vertical slice of a relation
- General form

$$\Pi_{A_1,\ldots,A_n}(R)=\{t[A_1,\ldots, A_n] \mid t\in R\}$$

  where
  - R is a relation, t is a tuple variable
  - $\{A_1,\ldots, A_n\}$ is a subset of the attributes of R over which the projection will be performed
- Note: projection can generate duplicate tuples. Commercial systems (and SQL) allow this and provide
  - Projection with duplicate elimination
  - Projection without duplicate elimination

Projection Example

PROJ

PNO	PNAME	BUDGET
P1	Instrumentation	150000
P2	Database Develop.	135000
P3	CAD/CAM	250000
P4	Maintenance	310000
P5	CAD/CAM	500000

$\Pi_{PNO,BUDGET}(PROJ)$

PNO	BUDGET
P1	150000
P2	135000
P3	250000
P4	310000
P5	500000

Cartesian (Cross) Product

- Given relations
  - R of degree k_1, cardinality n_1
  - S of degree k_2, cardinality n_2
- Cartesian (cross) product:

$$R \times S = \{t\ [A_1,\ldots,A_{k_1}, A_{k_1+1},\ldots,A_{k_1+k_2}] \mid t[A_1,\ldots,A_{k_1}] \in R \text{ and } t[A_{k_1+1},\ldots,A_{k_1+k_2}] \in S\}$$

The result of $R \times S$ is a relation of degree $(k_1 + k_2)$ and consists of all $(n_1 * n_2)$-tuples where each tuple is a concatenation of one tuple of R with one tuple of S.

Cartesian Product Example

EMP

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng
E2	M. Smith	Syst. Anal.
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E5	B. Casey	Syst. Anal.
E6	L. Chu	Elect. Eng.
E7	R. Davis	Mech. Eng.
E8	J. Jones	Syst. Anal.

PAY

TITLE	SALARY
Elect. Eng.	55000
Syst. Anal.	70000
Mech. Eng.	45000
Programmer	60000

EMP × PAY

ENO	ENAME	EMP.TITLE	PAY.TITLE	SALARY
E1	J. Doe	Elect. Eng.	Elect. Eng.	55000
E1	J. Doe	Elect. Eng.	Syst. Anal.	70000
E1	J. Doe	Elect. Eng.	Mech. Eng.	45000
E1	J. Doe	Elect. Eng.	Programmer	60000
E2	M. Smith	Syst. Anal.	Elect. Eng.	55000
E2	M. Smith	Syst. Anal.	Syst. Anal.	70000
E2	M. Smith	Syst. Anal.	Mech. Eng.	45000
E2	M. Smith	Syst. Anal.	Programmer	60000
E3	A. Lee	Mech. Eng.	Elect. Eng.	55000
E3	A. Lee	Mech. Eng.	Syst. Anal.	70000
E3	A. Lee	Mech. Eng.	Mech. Eng.	45000
E3	A. Lee	Mech. Eng.	Programmer	60000
E8	J. Jones	Syst. Anal.	Elect. Eng.	55000
E8	J. Jones	Syst. Anal.	Syst. Anal.	70000
E8	J. Jones	Syst. Anal.	Mech. Eng.	45000
E8	J. Jones	Syst. Anal.	Programmer	60000

Intersection

- Typical set intersection

$$R \cap S = \{t \mid t \in R \text{ and } t \in S\}$$

$$= R - (R - S)$$

- *R*, *S* union-compatible

Join

- General form

  $$R \bowtie_{F(R.A_i, S.B_j)} S = \{t[A_1,\ldots,A_n,B_1,\ldots,B_m] \mid t[A_1,\ldots,A_n] \in R \text{ and } t[B_1,\ldots,B_m] \in S \text{ and } F(R.A_i, S.B_j) \text{ is true}\}$$

  where

  - R, S are relations, t is a tuple variable
  - $F(R.A_i, S.B_j)$ is a formula defined as that of selection.

- A derivative of Cartesian product
  - $R \bowtie_F S = \sigma_F(R \times S)$

Join Example

EMP

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng
E2	M. Smith	Syst. Anal.
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E5	B. Casey	Syst. Anal.
E6	L. Chu	Elect. Eng.
E7	R. Davis	Mech. Eng.
E8	J. Jones	Syst. Anal.

WORKS

ENO	PNO	RESP	DUR
E1	P1	Manager	12
E2	P1	Analyst	24
E2	P2	Analyst	6
E3	P3	Consultant	10
E3	P4	Engineer	48
E4	P2	Programmer	18
E5	P2	Manager	24
E6	P4	Manager	48
E7	P3	Engineer	36
E7	P5	Engineer	23
E8	P3	Manager	40

$EMP \bowtie_{EMP.ENO>WORKS.ENO} WORKS$

EMP. ENO.	ENAME	TITLE	WORKS. ENO	PNO	RESP	DUR
E2	M. Smith	Elect. Eng.	E1	P1	Manager	12
E3	A. Lee	Syst. Anal.	E1	P1	Manager	12
E3	A. Lee	Syst. Anal.	E2	P1	Analyst	24
E3	A. Lee	Syst. Anal.	E2	P2	Analyst	6
E4	J. Miller	Programmer	E1	P1	Manager	12
E4	J. Miller	Programmer	E2	P1	Analyst	24
E4	J. Miller	Programmer	E2	P2	Analyst	6
E4	J. Miller	Programmer	E3	P3	Consultant	10
E4	J. Miller	Programmer	E3	P4	Engineer	48
E5	B. Casey	Syst. Anal.	E1	P1	Manager	12
E5	B. Casey	Syst. Anal.	E2	P1	Analyst	24
E5	B. Casey	Syst. Anal.	E2	P2	Analyst	6
E5	B. Casey	Syst. Anal.	E3	P3	Consultant	10
E5	B. Casey	Syst. Anal.	E3	P4	Engineer	48
E5	B. Casey	Syst. Anal.	E4	P2	Programmer	18
E6	L. Chu	Elect. Eng.	E1	P1	Manager	12
E6	L. Chu	Elect. Eng.	E2	P1	Analyst	24
E6	L. Chu	Elect. Eng.	E2	P2	Analyst	6
E6	L. Chu	Elect. Eng.	E3	P3	Consultant	10
E6	L. Chu	Elect. Eng.	E3	P4	Engineer	48
E6	L. Chu	Elect. Eng.	E4	P2	Programmer	18
E6	L. Chu	Elect. Eng.	E5	P2	Manager	24
…	…	…	…	…	…	…

Types of Join

- θ-join
  - The formula *F* uses operator θ
- Equi-join
  - The formula *F* only contains equality
  - $R \bowtie_{R.A=S.B} S$
- Natural join
  - Equi-join of two relations *R* and *S* over an attribute (or attributes) common to both *R* and *S* and projecting out one copy of those attributes
  - $R \bowtie S = \Pi_{R \cup S} \sigma_F (R \times S)$

Natural Join Example

EMP

ENO	ENAME	TITLE
E1	J. Doe	Elect. Eng
E2	M. Smith	Syst. Anal.
E3	A. Lee	Mech. Eng.
E4	J. Miller	Programmer
E5	B. Casey	Syst. Anal.
E6	L. Chu	Elect. Eng.
E7	R. Davis	Mech. Eng.
E8	J. Jones	Syst. Anal.

PAY

TITLE	SALARY
Elect. Eng.	55000
Syst. Anal.	70000
Mech. Eng.	45000
Programmer	60000

EMP $\bowtie$ PAY

ENO	ENAME	TITLE	SALARY
E1	J. Doe	Elect. Eng.	55000
E2	M. Smith	Analyst	70000
E3	A. Lee	Mech. Eng.	45000
E4	J. Miller	Programmer	60000
E5	B. Casey	Syst. Anal.	70000
E6	L. Chu	Elect. Eng.	55000
E7	R. Davis	Mech. Eng.	45000
E8	J. Jones	Syst. Anal.	70000

Join is over the common attribute TITLE

Types of Join

- Outer-Join
  - Ensures that tuples from one or both relations that do not satisfy the join condition still appear in the final result with other relation's attribute values set to NULL
  - Left outer join $⟕$
  - Right outer join $⟖$
  - Full outer join $⟗$

Division (Quotient)

Given relations

- R of degree k_1 ($R = \{A_1,\ldots,A_{k_1}\}$)
- S of degree k_2 ($S = \{B_1,\ldots,B_{k_2}\}$)

Let $A = \{A_1,\ldots,A_{k_1}\}$ [i.e., $R(A)$]and $B = \{B_1,\ldots,B_{k_2}\}$ [i.e., $S(B)$] and $B \subseteq A$.

Then, $T = R \div S$ gives T of degree k_1-k_2 [i.e., $T(Y)$ where $Y = A$-B] such that for a tuple t to appear in T, the values in t must appear in R in combination with *every tuple* in S.

Division (cont'd)

R

X	Y
x1	y1
x2	y1
x3	y1
x4	y1
x1	y2
x3	y2
x2	y3
x3	y3
x4	y3
x1	y4
x2	y4
x3	y4

S

X
x1
x2
x3

T

Y
y1
y4

$T_1 \leftarrow \Pi_Y(R)$

$T_2 \leftarrow \Pi_Y((S \times T_1) - R)$

$T \leftarrow T_1 - T_2$

Division - Formally

Given relations

- R of degree k_1 ($R = \{A_1,\ldots,A_{k_1}\}$)
- S of degree k_2 ($S = \{B_1,\ldots,B_{k_2}\}$)

Division of R by S (given , $\{B_1,\ldots,B_{k_2}\} \subseteq \{A_1,\ldots,A_{k_1}\}$)

$$R \div S = \{t[\{A_1,\ldots,A_{k_1}\} - \{B_1,\ldots,B_{k_2}\}] \mid \\ \forall u \in S \exists v \in R (v[S]=u \wedge v[R-S]=t)\} = \\ \Pi_{R\text{-}S}(R) - \Pi_{R\text{-}S}((\Pi_{R\text{-}S}(R) \times S) - R)$$

$R \div S$ results in a relation of degree $(k_1 - k_2)$ and consists of all $(k_1 - k_2)$-tuples t such that for all k_1-tuples u in S, the tuple tu is in R.

Division Example

EMP

ENO	PNO	PNAME	BUDGET
E1	P1	Instrumentation	150000
E2	P1	Instrumentation	150000
E2	P2	Database Develop.	135000
E3	P1	Instrumentation	150000
E3	P4	Maintenance	310000
E4	P2	Instrumentation	150000
E5	P2	Instrumentation	150000
E6	P4	Maintenance	310000
E7	P3	CAD/CAM	250000
E8	P3	CAD/CAM	250000
E3	P2	Database Develop.	135000
E3	P3	CAD/CAM	250000

PROJ

PNO	PNAME	BUDGET
P1	Instrumentation	150000
P2	Database Develop.	135000
P3	CAD/CAM	250000
P4	Maintenance	310000

EMP÷PROJ

ENO
E3

Example Queries

Emp (<u>Eno</u>, Ename, Title, City) (note we added City)
Project(<u>Pno</u>, Pname, Budget, City) (note we added City)
Pay(<u>Title</u>, Salary)
Works(<u>Eno, Pno</u>, Resp, Dur)

- List names of all employees.
  - $\Pi_{Ename}(Emp)$
- List names of all projects together with their budgets.
  - $\Pi_{Pname,Budget}(Project)$

Example Queries

Emp (Eno, Ename, Title, City) (note we added City)
Project(Pno, Pname, Budget, City) (note we added City)
Pay(Title, Salary)
Works(Eno, Pno, Resp, Dur)

- Find all job titles to which at least one employee has been hired.
  - $\Pi_{Title}(Emp)$
- Find the records of all employees who work in Toronto.
  - $\sigma_{City=\text{'Toronto'}}(Emp)$

Example Queries

Emp (Eno, Ename, Title, City)
Project(Pno, Pname, Budget, City)
Pay(Title, Salary)
Works(Eno, Pno, Resp, Dur)

- Find all cities where either an employee works or a project exists.
  - $\Pi_{City}(Emp) \cup \Pi_{City}(Project)$
- Find all cities that has a project but no employees who work there.
  - $\Pi_{City}(Project) - \Pi_{City}(Emp)$

Example Queries

Emp (Eno, Ename, Title, City)
Project(Pno, Pname, Budget, City)
Pay(Title, Salary)
Works(Eno, Pno, Resp, Dur)

- Find the names of all projects with budgets greater than $225,000.
  - $\Pi_{Pname}(\sigma_{Budget>225000}(Project))$
- List the names and budgets of projects on which employee E1 works.
  - $\Pi_{Pname, Budget}(Project \bowtie (\sigma_{Eno='E1'}(Works)))$
  - $\Pi_{Pname, Budget}(\sigma_{Emp.Eno=Works.Eno}(Project \times \sigma_{Eno='E1'}(Works)))$

Example Queries

Emp (<u>Eno</u>, Ename, Title, City)
Project(<u>Pno</u>, Pname, Budget, City)
Pay(<u>Title</u>, Salary)
Works(<u>Eno, Pno</u>, Resp, Dur)

- Find the name of all the employees who work in a city where no project is located.
  - $\Pi_{Ename}(Emp \bowtie (\Pi_{City}(Emp) - \Pi_{City}(Project))$
- Find all the cities that have both employees and projects.
  - $\Pi_{City}(Emp) \cap \Pi_{City}(Project)$
- Find all the employees who work on every project.
  - $\Pi_{Eno,\ Pno}(Works) \div \Pi_{Pno}(Project)$

Example Queries

Emp (<u>Eno</u>, Ename, Title, City)
Project(<u>Pno</u>, Pname, Budget, City)
Pay(<u>Title</u>, Salary)
Works(<u>Eno, Pno</u>, Resp, Dur)

- Find the names and budgets of all projects who employ programmers.
  - $\Pi_{Pname,Budget}(Project \bowtie Works \bowtie \sigma_{Title=\text{'Programmer'}}(Emp))$
- List the names of employees and projects that are co-located.
  - $\Pi_{Ename,\ Pname}(Emp \bowtie Project)$

Relational Calculus

- Instead of specifying how to obtain the result, specify what the result is, i.e., the relationships that is supposed to hold in the result.
- Based on first-order predicate logic.
  - symbol alphabet
    - logic symbols (e.g., $\Rightarrow$, $\neg$)
    - a set of constants
    - a set of variables
    - a set of n-ary predicates
    - a set of n-ary functions
    - parentheses
  - expressions (called well formed formulae (wff)) built from this symbol alphabet.

Types of Relational Calculus

- According to the primitive variable used in specifying the queries.
  - tuple relational calculus
  - domain relational calculus

Tuple Relational Calculus

- The primitive variable is a tuple variable which specifies a tuple of a relation. In other words, it ranges over the tuples of a relation.
- In tuple relational calculus queries are specified as

  $$\{t \mid F(t)\}$$

  where t is a tuple variable and F is a formula consisting of the atoms and operators. F evaluates to True or False.

  t can be qualified for only some attributes: $t[A]$

Tuple Relational Calculus

- The atoms are the following:
  1. Tuple variables
     - If the relation over which the variable ranges is known, the variable may be qualified by the name of the relation as *R*.*t* or *R*(*t*).
  2. Conditions
     - $s[A]\ \theta\ t[B]$, where *s* and *t* are tuple variables and *A* and *B* are components of *s* and *t*, respectively;

       $\theta \in \{<, >, =, \neq, \leq, \geq\}$.

       Specifies that component *A* of *s* stands in relation θ to the *B* component of *t* (e.g., *s*[SALARY] > *t*[SALARY]).
     - $s[A]\ \theta\ c$, where *s*, *A* and θ are as defined above and *c* is a constant. For example, *s*[NAME] = "Smith".

Tuple Relational Calculus

- A formula F is composed of
  - atoms
  - Boolean operators $\wedge$, $\vee$, $\neg$
  - existential quantifier $\exists$
  - universal quantifier $\forall$
- Formation rules:
  - Each atom is a formula.
  - If F and G are formulae, so are $F \wedge G$, $F \vee G$, $\neg F$, and $\neg G$.
  - If F is a formula, so is (F).
  - If F is a formula and t is a free variable in F, then $\exists t(F)$ and $\forall t(F)$ are also formulae. These can also be written as $\exists t F(t)$ and $\forall t F(t)$
  - Nothing else is a formula.

Safety of Calculus Expressions

- Problem:
  - the size of $\{t \mid F(t)\}$ must be finite.
  - $\{t \mid \neg t \in R\}$ is not finite
- Safety:
  - A query is safe if, for all databases conforming to the schema, the query result can be computed using only constants appearing in the database or the query itself.
  - Since database is finite, the set of constants appearing in it is finite as well as the constants in the query; therefore, the query result will be finite

Example Queries

Emp (Eno, Ename, Title, City)
Project(Pno, Pname, Budget, City)
Pay(Title, Salary)
Works(Eno, Pno, Resp, Dur)

- List names of all employees.

  $\{t[\text{Ename}] \mid t \in \text{Emp}\}$

- List names of all projects together with their budgets.

  $\{<t[\text{Pname}],t[\text{Budget}]> \mid t \in \text{Project}]\}$

Example Queries

Emp (<u>Eno</u>, Ename, Title, City)
Project(<u>Pno</u>, Pname, Budget, City)
Pay(<u>Title</u>, Salary)
Works(<u>Eno, Pno</u>, Resp, Dur)

- Find all job titles to which at least one employee has been hired.

  $\{t[\text{Title}] | t \in \text{Emp})\}$

- Find the records of all employees who work in Toronto.

  $\{t \mid t \in \text{customer} \wedge t[\text{City}] = \text{'Toronto'}\}$

Example Queries

Emp (Eno, Ename, Title, City)
Project(Pno, Pname, Budget, City)
Pay(Title, Salary)
Works(Eno, Pno, Resp, Dur)

- Find all cities where either an employee works or a project exists.

  $\{t[\text{City}] \mid t \in \text{Emp} \vee \exists s(s \in \text{Project} \wedge t[\text{City}] = s[\text{City}])\}$

  $\{t[\text{City}] \mid t \in \text{Project} \vee \exists s(s \in \text{Emp} \wedge t[\text{City}] = s[\text{City}])\}$

- Find all cities that has a project but no employees who work there.

  $\{t[\text{City}] \mid t \in \text{Project} \wedge \neg\exists s(s \in \text{Emp} \wedge t[\text{City}] = s[\text{City}])\}$

Example Queries

Emp (<u>Eno</u>, Ename, Title, City)
Project(<u>Pno</u>, Pname, Budget, City)
Pay(<u>Title</u>, Salary)
Works(<u>Eno, Pno</u>, Resp, Dur)

- Find the names of all projects with budgets greater than $225,000.

  $\{t[\text{Pname}] \mid t \in \text{Project} \wedge t[\text{Budget}] > 225000\}$

- List the names and budgets of projects in which employee E1 works.

  $\{<t[\text{Pname}], t[\text{Budget}]> \mid$
  $t \in \text{Project} \wedge \exists s(s \in \text{Works} \wedge t[\text{Pname}] = s[\text{Pname}] \wedge$
  $s[\text{Eno}] = \text{'E1'})\}$

Tuple Calculus and Relational Algebra

- THEOREM: Safe relational calculus and algebra are equivalent in terms of their expressive power.
  - This is called relational completeness.
  - This does not mean all useful computations can be performed
    - Aggregation, counting, transitive closure not specified
- Basic Correspondence

Algebra Operation	Calculus Operator
Π	$\exists$
σ	$x\ \theta\ constant$
$\cup$	$\vee$
$\bowtie$	$\wedge$
-	$\neg$
$\div$	$\forall$

Tuple Calculus and Relational Algebra

Π is like $\exists$ "there exists" ...

$\div$ is like $\forall$ "for all" ...

Expressing $\div$ using basic operators

$$R \div S = \Pi_A(R) - \Pi_A(\Pi_A(R) \bowtie S - R)$$

Similar to

$$\forall x\, F(x) = \neg \quad (\exists x \quad \neg F(x))$$

Domain Relational Calculus

- The primitive variable is a domain variable which specifies a component of a tuple.

  ⇨ the range of a domain variable consists of the domains over which the relation is defined.
- Other differences from tuple relational calculus:
  - The atoms are the following :
    - Each domain is an atom.
    - Conditions which can be defined as follows are atoms :
      - $x \theta y$, where x and y are domain variables or constants;
      - $<x_1, x_2, ..., x_n> \in R$ where R is a relation of degree n and each x_i is a domain variable or constant.
  - Formulae are defined in exactly the same way as in tuple relational calculus, with the exception of using domain variables instead of tuple variables.

Domain Relational Calculus

The queries are specified in the following form :

$$\{<x_1, x_2, ..., x_n> \mid F(x_1, x_2, ..., x_n)\}$$

where F is a formula in which $x_1,..., x_n$ are the free variables.

Domain Relational Calculus

Emp (Eno, Ename, Title, City)
Project(Pno, Pname, Budget, City)
Pay(Title, Salary)
Works(Eno, Pno, Resp, Dur)

List the names and budgets of projects in which employee E1 works.

$$\{<b,c> \mid \exists a,d(<a,b,c,d> \in \text{Project} \wedge \exists e,f,g(<e,a,f,g> \in \text{Works} \wedge e = \text{'E1'}))\}$$

QBE (Query-by-Example) Queries

- Find the names of all employees

Emp	Eno	Ename	Title	City
		P.		

Pay	Title	Salary

Project	Pno	Pname	Budget	City

Works	Eno	Pno	Resp	Dur

QBE Queries

- Find the names of projects with budgets greater than $350,000.

Emp	Eno	Ename	Title	City

Pay	Title	Salary

Project	Pno	Pname	Budget	City
		P.	>350000	

Works	Eno	Pno	Resp	Dur

QBE Queries

- Find the name and cities of all employees who work on a project for more than 20 months.

Emp	Eno	Ename	Title	City
	_X	P.		P.

Pay	Title	Salary

Project	Pno	Pname	Budget	City

Works	Eno	Pno	Resp	Dur
	_X			>20