Chapter 2: Intro to Relational Model

- Attribute, domain, tuple
- Relation, schema, instance
- Keys: foreign key, primary key, referential integrity
- Relational query languages
- Basic relational operators

Chapter 6: Formal Relational Query Languages

- Relational algebra
  - Basic operators
  - Additional operators
  - Extended algebra (generalized projection and aggregation))
- Tuple relational calculus
- Domain relational calculus

Example of a Relation

attributes a_i
(or columns)

ID	*name*	*dept_name*	*salary*
10101	Srinivasan	Comp. Sci.	65000
12121	Wu	Finance	90000
15151	Mozart	Music	40000
22222	Einstein	Physics	95000
32343	El Said	History	60000
33456	Gold	Physics	87000
45565	Katz	Comp. Sci.	75000
58583	Califieri	History	62000
76543	Singh	Finance	80000
76766	Crick	Biology	72000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000

tuples t_i
(or rows)

Attribute Types

- The set of allowed values for each attribute is called the **domain** of the attribute
- Attribute values are (normally) required to be **atomic**; that is, indivisible
- The special value ***null*** is a member of every domain
- The null value causes complications in the definition of many operations

Relation Schema and Instance

- $A_1, A_2, \ldots, A_n$ are *attributes*

- $R = (A_1, A_2, \ldots, A_n)$ is a *relation schema*

  Example: *instructor* = (*ID*, *name*, *dept_name*, *salary*)

- Formally, given sets $D_1, D_2, \ldots. D_n$ a **relation** ***r*** is a subset of

  $$D_1 \times D_2 \times \ldots \times D_n$$

  Thus, a relation is a set of n-tuples $(a_1, a_2, \ldots, a_n)$ where each $a_i \in D_i$

- The current values (**relation instance**) of a relation are specified by a table

- An element ***t*** of ***r*** is a *tuple*, represented by a *row* in a table

Relations are Unordered

- Order of tuples is irrelevant (tuples may be stored in an arbitrary order)
- Example: *instructor* relation with unordered tuples

ID	*name*	*dept_name*	*salary*
22222	Einstein	Physics	95000
12121	Wu	Finance	90000
32343	El Said	History	60000
45565	Katz	Comp. Sci.	75000
98345	Kim	Elec. Eng.	80000
76766	Crick	Biology	72000
10101	Srinivasan	Comp. Sci.	65000
58583	Califieri	History	62000
83821	Brandt	Comp. Sci.	92000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
76543	Singh	Finance	80000

Database

- A database consists of multiple relations
- Information about an enterprise is broken up into parts

  instructor
  student
  advisor

- Bad design:

  university (*instructor -ID, name, dept_name, salary, student_Id*, ..)

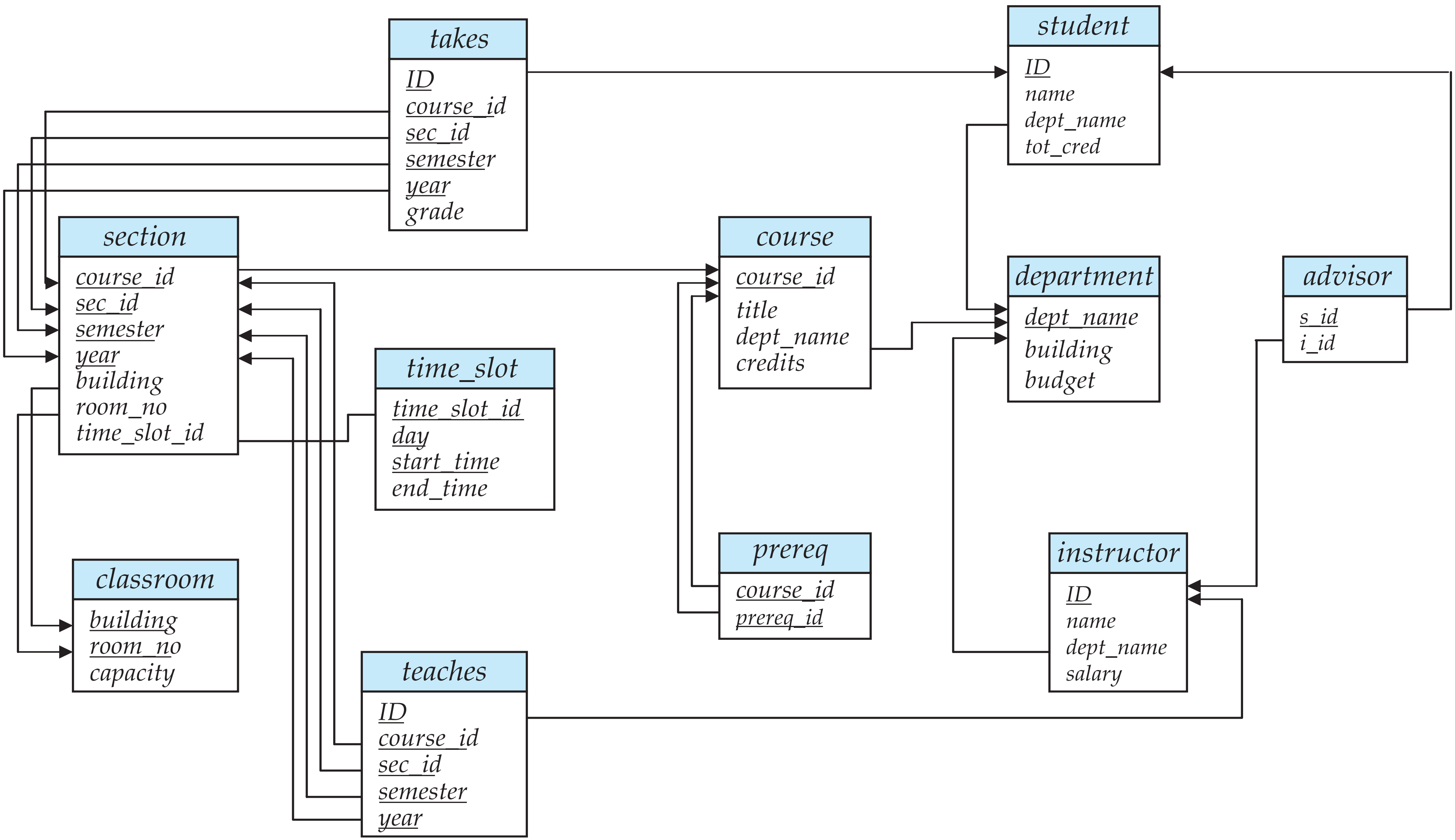
  results in
  - repetition of information (e.g., two students have the same instructor)
  - the need for null values (e.g., represent an student with no advisor)
- Normalization theory (Chapter 7) deals with how to design “good” relational schemas

Keys

- Let K $\subseteq$ R, R set of attributes of a relation schema
- *K* is a **superkey** of *R* if values for *K* are sufficient to identify a unique tuple of each possible relation *r(R)*
  - Example: {*ID*} and {ID,name} are both superkeys of *instructor.*
- Superkey *K* is a **candidate key** if *K* is minimal

  Example: {*ID*} is a candidate key for *Instructor*
- One of the candidate keys is selected to be the **primary key**.
  - which one?
- **Foreign key** constraint: Value in one relation must appear in another
  - **Referencing** relation
  - **Referenced** relation

Schema Diagram for University Database



Relational Query Languages

- Procedural vs.non-procedural, or declarative
- “Pure” languages:
  - Relational algebra
  - Tuple relational calculus
  - Domain relational calculus
- Relational operators

Selection of tuples

- Relation r

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

- Select tuples with A=B and D > 5
  - $\sigma_{A=B \text{ and } D > 5}(r)$

A	B	C	D
α	α	1	7
β	β	23	10

Selection of Columns (Attributes)

- Relation *r*:

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

- Select A and C
  - Projection
  - $\Pi_{A,\ C}$ (r)

A	C
α	1
α	1
β	1
β	2

=

A	C
α	1
β	1
β	2

Joining two relations – Cartesian Product

- Relations *r, s*:

A	B
α	1
β	2

r

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

s

- *r* x *s*:

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

Union of two relations

- Relations *r, s:*

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- r ∪ s:

A	B
α	1
α	2
β	1
β	3

Set difference of two relations

- Relations *r*, *s*:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- $r - s$:

A	B
α	1
β	1

Set Intersection of two relations

- Relation *r, s*:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- $r \cap s$

A	B
α	2

Joining two relations – Natural Join

- Let r and s be relations on schemas R and S respectively. Then, the "natural join" of relations R and S is a relation on schema $R \cup S$ obtained as follows:
  - Consider each pair of tuples t_r from r and t_s from s.
  - If t_r and t_s have the same value on each of the attributes in $R \cap S$, add a tuple t to the result, where
    - t has the same value as t_r on r
    - t has the same value as t_s on s

Natural Join Example

- Relations r, s:

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

r

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	ε

s

- Natural Join
  - r $\bowtie$ s

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

Overview of Operators (see page 51)

Symbol (Name)	Example of Use
σ (Selection)	$\sigma_{\text{salary}>=85000}(instructor)$
	Return rows of the input relation that satisfy the predicate.
Π (Projection)	$\Pi_{ID,\ salary}(instructor)$
	Output specified attributes from all rows of the input relation. Remove duplicate tuples from the output.
⋈ (Natural Join)	$instructor \bowtie department$
	Output pairs of rows from the two input relations that have the same value on all attributes that have the same name.
× (Cartesian Product)	$instructor \times department$
	Output all pairs of rows from the two input relations (regardless of whether or not they have the same values on common attributes)
∪ (Union)	$\Pi_{name}(instructor) \cup \Pi_{name}(student)$
	Output the union of tuples from the two input relations.

Relational Algebra

- Procedural language
- Based on relational operations introduced in chapter 2
- Six basic operators
  - select: σ
  - project: $\prod$
  - union: $\cup$
  - set difference: –
  - cartesian product: x
  - rename: ρ
- The operators take one or two relations as inputs and produce a new relation as a result
- Additional operators defined on top of these 6 later

Select Operation – Example

- Relation r

A	B	C	D
α	α	1	7
α	β	5	7
β	β	12	3
β	β	23	10

- $\sigma_{A=B \wedge D > 5}(r)$

A	B	C	D
α	α	1	7
β	β	23	10

Select Operation

- Notation: $\sigma_p(r)$
- *p* is called the **selection predicate**
- Defined as:

$$\sigma_p(\boldsymbol{r}) = \{t \mid t \in r \textbf{ and } p(t)\}$$

Where *p* is a formula in propositional calculus consisting of **terms** connected by : $\wedge$ (**and**), $\vee$ (**or**), $\neg$ (**not**)
Each **term** is one of:

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

where *op* is one of: $=, \neq, >, \geq. <. \leq$

- Example of selection:

$$\sigma_{dept\_name=\text{“}Physics\text{”}}(instructor)$$

Project Operation – Example

- Relation *r*:

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

- $\prod_{A,C}(r)$

A	C
α	1
α	1
β	1
β	2

=

A	C
α	1
β	1
β	2

Project Operation

- Notation: $$\prod_{A_1, A_2, \ldots, A_k} (r)$$

  where A_1, A_2 are attribute names and r is a relation name.
- The result is defined as the relation of k columns obtained by erasing the columns that are not listed
- Duplicate rows removed from result, since relations are sets
- Example: To eliminate the *dept_name* attribute of *instructor*

$$\prod_{ID,\ name,\ salary} (instructor)$$

Union Operation – Example

- Relations *r, s:*

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- r ∪ s:

A	B
α	1
α	2
β	1
β	3

Set difference of two relations

- Relations *r*, *s*:

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- $r - s$:

A	B
α	1
β	1

Cartesian-Product Operation – Example

- Relations *r*, *s*:

A	B
α	1
β	2

r

C	D	E
α	10	a
β	10	a
β	20	b
γ	10	b

s

- *r* x *s*:

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

Cartesian-Product Operation

- Notation $r \times s$
- Defined as:

$$r \times s = \{t\,q \mid t \in r \textbf{ and } q \in s\}$$

- Assume that attributes of r(R) and s(S) are disjoint. (That is, $R \cap S = \emptyset$).
- If attributes of *r(R)* and *s(S)* are not disjoint, then renaming must be used.

Composition of Operations

- Can build expressions using multiple operations
- Example: $\sigma_{A=C}(r \times s)$
- $r \times s$

A	B	C	D	E
α	1	α	10	a
α	1	β	10	a
α	1	β	20	b
α	1	γ	10	b
β	2	α	10	a
β	2	β	10	a
β	2	β	20	b
β	2	γ	10	b

- $\sigma_{A=C}(r \times s)$

A	B	C	D	E
α	1	α	10	a
β	2	β	10	a
β	2	β	20	b

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:

$$\rho_X(E)$$

returns the expression *E* under the name *X*

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

$$\rho_{x(A_1, A_2, \ldots, A_n)}(E)$$

returns the result of expression *E* under the name *X*, and with the attributes renamed to $A_1, A_2, \ldots, A_n$.

Formal Definition

- A basic expression in the relational algebra consists of either one of the following:
  - A relation in the database
  - A constant relation
- Let E_1 and E_2 be relational-algebra expressions; the following are all relational-algebra expressions:
  - $E_1 \cup E_2$
  - $E_1 - E_2$
  - $E_1 \times E_2$
  - $\sigma_p (E_1)$, P is a predicate on attributes in E_1
  - $\prod_S(E_1)$, S is a list consisting of some of the attributes in E_1
  - $\rho_{x} (E_1)$, x is the new name for the result of E_1

Additional Operations

We define additional operations that do not add any power to the relational algebra, but that simplify common queries.

- Set intersection
- Natural join
- Assignment
- Outer join

Set-Intersection Operation

- Notation: $r \cap s$
- Defined as:
- $r \cap s = \{ t \mid t \in r \textbf{ and } t \in s \}$
- Assume:
  - *r*, *s* have the *same arity*
  - attributes of *r* and *s* are compatible
- Note: $r \cap s = r - (r - s)$
- Example
  - Relations r, s

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

  - $r \cap s$

A	B
α	2

Natural Join Operation

- Notation: $r \bowtie s$
- Let *r* and *s* be relations on schemas *R* and *S* respectively. Then, $r \bowtie s$ is a relation on schema $R \cup S$ obtained as follows:
  - Consider each pair of tuples t_r from *r* and t_s from *s*.
  - If t_r and t_s have the same value on each of the attributes in $R \cap S$, add a tuple *t* to the result, where
    - *t* has the same value as t_r on *r*
    - *t* has the same value as t_s on *s*

Natural Join Example

- Example:

  $R = (A, B, C, D)$

  $S = (E, B, D)$

  - Result schema = (A, B, C, D, E)
  - r s is defined as:

    $$\Pi_{r.A,\ r.B,\ r.C,\ r.D,\ s.E}\ (\sigma_{r.B = s.B \wedge r.D = s.D}\ (r \times s))$$

- Relations r, s:

A	B	C	D
α	1	α	a
β	2	γ	a
γ	4	β	b
α	1	γ	a
δ	2	β	b

r

B	D	E
1	a	α
3	a	β
1	a	γ
2	b	δ
3	b	ε

s

- r $\bowtie$ s

A	B	C	D	E
α	1	α	a	α
α	1	α	a	γ
α	1	γ	a	α
α	1	γ	a	γ
δ	2	β	b	δ

Natural Join and Theta Join

- Find the names of all instructors in the Comp. Sci. department together with the course titles of all the courses that the instructors teach
  - $\prod_{name,\ title} (\sigma_{dept\_name = \text{“Comp. Sci.”}} (instructor \bowtie teaches \bowtie course))$
- Natural join is associative
  - $(instructor \bowtie teaches) \bowtie course$ is equivalent to $instructor \bowtie (teaches \bowtie course)$
- Natural join is commutative
  - $instructor \bowtie teaches$ is equivalent to $teaches \bowtie instructor$
- The **theta join** operation $r \bowtie_\theta s$ is defined as
  - $r \bowtie_\theta s = \sigma_\theta (r \times s)$

Assignment Operation

- The assignment operation ($\leftarrow$) 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:

$$temp1 \leftarrow instructor \bowtie teaches \bowtie course$$

$$temp2 \leftarrow \sigma_{dept\_name=\text{"Comp. Sci."}}(temp1)$$

$$\text{result} = \Pi_{name,\ title}(temp2)$$

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

Outer Join – Example

- Relation *instructor*

ID	*name*	*dept_name*
10101	Srinivasan	Comp. Sci.
12121	Wu	Finance
15151	Mozart	Music

- Relation *teaches*

ID	*course_id*
10101	CS-101
12121	FIN-201
76766	BIO-101

Outer Join – Example (Cont.)

- Natural Join

instructor $\bowtie$ *teaches*

ID	*name*	*dept_name*	*course_id*
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201

- Left Outer Join

instructor ⟕ *teaches*

ID	*name*	*dept_name*	*course_id*
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201
15151	Mozart	Music	*null*

Outer Join – Example (Cont.)

- Right Outer Join *instructor* ⟖ *teaches*

ID	*name*	*dept_name*	*course_id*
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201
76766	null	null	BIO-101

- Full Outer Join *instructor* ⟗ *teaches*

ID	*name*	*dept_name*	*course_id*
10101	Srinivasan	Comp. Sci.	CS-101
12121	Wu	Finance	FIN-201
15151	Mozart	Music	*null*
76766	null	null	BIO-101

- Outer join can be expressed using basic operations
  - e.g. r ⟕ s can be written as

$$(r \bowtie s) \cup (r - \Pi_R(r \bowtie s) \times \{(null, \ldots, null)\})$$

Null Values

- It is possible for tuples to have a null value, denoted by *null*, for some of their attributes
- *null* signifies an unknown value or that a value does not exist.
- The result of any arithmetic expression involving *null* is *null.*
- Aggregate functions simply ignore null values (as in SQL)
- For duplicate elimination and grouping, null is treated like any other value, and two nulls are assumed to be the same (as in SQL)

Null Values

- Comparisons with null values return the special truth value: *unknown*
  - If *false* was used instead of *unknown*, then *not (A < 5)* would not be equivalent to *A >= 5*
- Three-valued logic using the truth value *unknown*:
  - OR: (*unknown* **or** *true*) = *true*,
    (*unknown* **or** *false*) = *unknown*
    (*unknown* **or** *unknown*) = *unknown*
  - AND: (*true* **and** *unknown*) = *unknown,*
    (*false* **and** *unknown*) = *false,*
    (*unknown* **and** *unknown*) = *unknown*
  - NOT*:* (**not** *unknown*) = *unknown*
  - In SQL "*P* **is unknown**" evaluates to true if predicate *P* evaluates to *unknown*
- Result of select predicate is treated as *false* if it evaluates to *unknown*

Extended Relational Algebra Operations

- Generalized Projection
- Aggregate Functions

Generalized Projection

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

$$\prod_{F_1, F_2, \ldots, F_n}(E)$$

- *E* is any relational-algebra expression
- Each of F_1, F_2, …, F_n are are arithmetic expressions involving constants and attributes in the schema of *E*.
- Given relation *instructor(ID, name, dept_name,* salary) where salary is annual salary, get the same information but with monthly salary

$$\Pi_{ID,\ name,\ dept\_name,\ salary/12}\ (instructor)$$

Aggregate Functions and Operations

- **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

$$_{G_1, G_2, \ldots, G_n}\ \mathcal{G}_{\ F_1(A_1), F_2(A_2), \ldots, F_n(A_n)}(E)$$

  E is any relational-algebra expression

  - $G_1, G_2 \ldots, G_n$ is a list of attributes on which to group (can be empty)
  - Each F_i is an aggregate function
  - Each A_i is an attribute name

- Note: Some books/articles use γ instead of $\mathcal{G}$ (Calligraphic G)

Aggregate Functions and Operations (Cont.)

- Result of aggregation does not have a name
  - Can use rename operation to give it a name
  - For convenience, we permit renaming as part of aggregate operation

$$_{dept\_name}\ \mathcal{G}_{\,\textbf{avg}(salary)\ \textbf{as}\ avg\_sal}\ (instructor)$$

Aggregate Operation – Example

- Relation *r*:

A	*B*	*C*
α	α	7
α	β	7
β	β	3
β	β	10

- $\mathcal{G}_{\mathbf{sum(c)}}\,(\mathrm{r})$

sum(*c*)
27

Aggregate Operation – Example (Cont.)

- Find the average salary in each department

$$_{dept\_name}\mathcal{G}_{\textbf{avg}(salary)}\,(instructor)$$

ID	*name*	*dept_name*	*salary*
76766	Crick	Biology	72000
45565	Katz	Comp. Sci.	75000
10101	Srinivasan	Comp. Sci.	65000
83821	Brandt	Comp. Sci.	92000
98345	Kim	Elec. Eng.	80000
12121	Wu	Finance	90000
76543	Singh	Finance	80000
32343	El Said	History	60000
58583	Califieri	History	62000
15151	Mozart	Music	40000
33456	Gold	Physics	87000
22222	Einstein	Physics	95000

dept_name	*avg_salary*
Biology	72000
Comp. Sci.	77333
Elec. Eng.	80000
Finance	85000
History	61000
Music	40000
Physics	91000

Modification of the Database

- The content of the database may be modified using the following operations:
  - Deletion
  - Insertion
  - Updating

- All these operations can be expressed using the assignment operator

Multiset Relational Algebra

- Pure relational algebra removes all duplicates
  - e.g. after projection
- Multiset relational algebra retains duplicates, to match SQL semantics
  - SQL duplicate retention was initially for efficiency, but is now a feature
- Multiset relational algebra defined as follows
  - selection: has as many duplicates of a tuple as in the input, if the tuple satisfies the selection
  - projection: one tuple per input tuple, even if it is a duplicate
  - cross product: If there are m copies of $t1$ in r, and n copies of $t2$ in s, there are m x n copies of $t1.t2$ in r x s
  - Other operators similarly defined
    - E.g. union: $m + n$ *copies,* intersection: min(m, n) copies difference: min(0, $m - n$) copies

Tuple Relational Calculus

- A nonprocedural query language, where each query is of the form

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

- It is the set of all tuples *t* such that predicate *P* is true for *t*
- *t* is a *tuple variable*, $t[A]$ denotes the value of tuple *t* on attribute *A*
- $t \in r$ denotes that tuple *t* is in relation *r*
- *P* is a *formula* similar to that of the predicate calculus

Predicate Calculus Formula

1. Set of attributes and constants
2. Set of comparison operators: (e.g., $<$, $\leq$, $=$, $\neq$, $>$, $\geq$)
3. Set of connectives: and ($\wedge$), or ($\vee$), not ($\neg$)
4. Implication ($\Rightarrow$): x $\Rightarrow$ y, if x if true, then y is true

$$x \Rightarrow y \equiv \neg x \vee y$$

5. Set of quantifiers:
   - $\exists\, t \in r\, (Q\,(t\,)) \equiv$ "there exists" a tuple t in relation r such that predicate $Q\,(t)$ is true
   - $\forall t \in r\, (Q\,(t\,)) \equiv Q$(t) is true "for all" tuples t in relation r

Example Queries

- Find the *ID, name, dept_name, salary* for instructors whose salary is greater than $80,000

$$\{t \mid t \in instructor \land t[salary] > 80000\}$$

- As in the previous query, but output only the *ID* attribute value

$$\{t \mid \exists\, s \in \text{instructor}\ (t[ID] = s[ID] \land s[salary] > 80000)\}$$

Notice that a relation on schema (*ID*) is implicitly defined by the query

Example Queries (Cont.)

- Find the names of all instructors whose department is in the Watson building

$$\{t \mid \exists s \in instructor\ (t[name] = s[name]$$
$$\wedge\ \exists u \in department\ (u[dept\_name] = s[dept\_name] \text{ “}$$
$$\wedge\ u[building] = \text{“Watson”}))\}$$

- Find the set of all courses taught in the Fall 2009 semester, or in the Spring 2010 semester, or both

$$\{t \mid \exists s \in section\ (t[course\_id] = s[course\_id] \wedge$$
$$s[semester] = \text{“Fall”} \wedge s[\text{year}] = 2009)$$
$$\vee\ \exists u \in section\ (t\ [course\_id] = u[course\_id] \wedge$$
$$u[semester] = \text{“Spring”} \wedge u[\text{year}] = 2010)\}$$

Example Queries (Cont.)

- Find the set of all courses taught in the Fall 2009 semester, and in the Spring 2010 semester

$$\{t \mid \exists s \in section\ (t[course\_id] = s[course\_id] \land s[semester] = \text{“Fall”} \land s[\text{year}] = 2009) \land \exists u \in section\ (t[course\_id] = u[course\_id] \land u[semester] = \text{“Spring”} \land u[\text{year}] = 2010)\}$$

- Find the set of all courses taught in the Fall 2009 semester, but not in the Spring 2010 semester

$$\{t \mid \exists s \in section\ (t[course\_id] = s[course\_id] \land s[semester] = \text{“Fall”} \land s[\text{year}] = 2009) \land \neg \exists u \in section\ (t[course\_id] = u[course\_id] \land u[semester] = \text{“Spring”} \land u[\text{year}] = 2010)\}$$

Safety of Expressions

- It is possible to write tuple calculus expressions that generate infinite relations.
- For example, $\{ t \mid \neg\ t \in r \}$ results in an infinite relation if the domain of any attribute of relation *r* is infinite
- To guard against the problem, we restrict the set of allowable expressions to safe expressions.
- An expression $\{t \mid P(t)\}$ in the tuple relational calculus is *safe* if every component (value) of *t* appears in one of the relations, tuples, or constants that appear in *P*
  - NOTE: this is more than just a syntax condition.
    - E.g. $\{ t \mid t[A] = 5 \lor \textbf{true} \}$ is not safe --- it defines an infinite set with attribute values that do not appear in any relation or tuples or constants in *P*.

Domain Relational Calculus

- A nonprocedural query language equivalent in power to the tuple relational calculus
- Each query is an expression of the form:

$$\{ < x_1, x_2, \ldots, x_n > \mid P (x_1, x_2, \ldots, x_n)\}$$

  - $x_1, x_2, \ldots, x_n$ represent domain variables
  - P represents a formula similar to that of the predicate calculus

Example Queries

- Find the *ID, name, dept_name, salary* for instructors whose salary is greater than $80,000
  - $\{< i, n, d, s> \mid < i, n, d, s> \in instructor \wedge s > 80000\}$
- As in the previous query, but output only the *ID* attribute value
  - $\{< i> \mid < i, n, d, s> \in instructor \wedge s > 80000\}$
- Find the names of all instructors whose department is in the Watson building

$$\{< n > \mid \exists\, i, d, s\, (< i, n, d, s > \in instructor \\ \wedge \exists\, \text{b, a}\, (< d, b, a> \in department \wedge b = \text{“Watson”}))\}$$

Example Queries

- Find the set of all courses taught in the Fall 2009 semester, or in the Spring 2010 semester, or both

$$\{<c> \mid \exists\, a, s, y, b, r, t\ (<c, a, s, y, b, t> \in section\ \wedge\ s = \text{“Fall”} \wedge y = 2009\ )\ \vee \exists\, a, s, y, b, r, t\, (<c, a, s, y, b, t> \in section\,]\ \wedge\ s = \text{“Spring”} \wedge y = 2010)\}$$

This case can also be written as

$$\{<c> \mid \exists\, a, s, y, b, r, t\ (<c, a, s, y, b, t> \in section\ \wedge\ (\ (s = \text{“Fall”} \wedge y = 2009\ )\ \vee (s = \text{“Spring”} \wedge y = 2010))\}$$

- Find the set of all courses taught in the Fall 2009 semester, and in the Spring 2010 semester

$$\{<c> \mid \exists\, a, s, y, b, r, t\ (<c, a, s, y, b, t> \in section\ \wedge\ s = \text{“Fall”} \wedge y = 2009\ )\ \wedge \exists\, a, s, y, b, r, t\, (<c, a, s, y, b, t> \in section\,]\ \wedge\ s = \text{“Spring”} \wedge y = 2010)\}$$

Relative Expressive Power

- Safe TRC and DRC are as powerful as (basic) relational algebra
- Extended relational algebra is
  - More powerful than TRC/DRC
  - Useful for defining SQL semantics
  - Useful for designing SQL queries
  - Useful for understanding SQL execution
- Relational calculus is
  - Useful for designing SQL queries