Introduction to Database Management

Data Models

- Databases must represent:
  - the data itself (typically structured in some way)
  - associations between different data values
  - optionally, constraints on data values
- What kind of data can be modeled?
- What kinds of associations can be represented?
- The data model specifies:
  - what data can be stored (and sometimes how it is stored)
  - associations between different data values
  - what constraints can be enforced
  - how to access and manipulate the data

Relations

- Relations are basically tables of data
  - Each row represents a record in the relation
- A relational database is a set of relations
  - Each relation has a unique name in the database
- Each row in the table specifies a relationship between the values in that row
  - The account ID "A-307", branch name "Seattle", and balance "275" are all related to each other

acct_id	branch_name	balance
A-301	New York	350
A-307	Seattle	275
A-318	Los Angeles	550
. . .	. . .	. . .

The account relation

Relations and Attributes

- Each relation has some number of attributes
  - Sometimes called “columns”
- Each attribute has a domain
  - Specifies the set of valid values for the attribute
- The account relation:
  - 3 attributes
  - Domain of *balance* is the set of nonnegative integers
  - Domain of *branch_name* is the set of all valid branch names in the bank

acct_id	branch_name	balance
A-301	New York	350
A-307	Seattle	275
A-318	Los Angeles	550
. . .	. . .	. . .

account

Tuples and Attributes

- Each row is called a tuple
  - A fixed-size, ordered set of name-value pairs
- A tuple variable can refer to any valid tuple in a relation
- Each attribute in the tuple has a unique name
- Can also refer to attributes by index
  - Attribute 1 is the first attribute, etc.
- Example:
  - Let tuple variable *t* refer to first tuple in *account relation*
  - *t*[*balance*] = 350
  - *t*[2] = "New York"

acct_id	branch_name	balance
A-301	New York	350
A-307	Seattle	275
A-318	Los Angeles	550
. . .	. . .	. . .

account

Tuples and Relations

- A relation is a set of tuples
  - Each tuple appears exactly once
    - *Note: SQL tables are multisets! (Sometimes called bags.)*
  - If two tuples t_1 and t_2 have the same values for all attributes, then t_1 and t_2 are the same tuple (i.e. $t_1 = t_2$)
- The order of tuples in a relation is not relevant

Example of a Relation

attributes
(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
(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
  - Indicated that the value is "unknown"
- The null value causes complications in the definition of many operations

Relation Schemas

- Every relation has a schema
  - Specifies the type information for relations
  - Multiple relations can have the same schema
- A relation schema includes:
  - an ordered set of attributes
  - the domain of each attribute
- Naming conventions:
  - Relation names are written as all lowercase
  - Relation schema's name is capitalized
- For a relation r and relation schema R:
  - Write $r(R)$ to indicate that the schema of r is R

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 $a_i \in D_i$

Relation Schema and Instance (cont.)

- 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

Schema of account Relation

- The relation schema of account is:

  Account_schema = (*acct_id*, *branch_name*, *balance*)

- To indicate that account has schema *Account_schema*:

  account(*Account_schema*)

acct_id	branch_name	balance
A-301	New York	350
A-307	Seattle	275
A-318	Los Angeles	550
. . .	. . .	. . .

account

- Important note:
  - Domains are not stated explicitly in this notation!

Relation Schemas

- Relation schemas are ordered sets of attributes
  - Can use set operations on them
- Examples:

  Relations $r(R)$ and $s(S)$
  - Relation r has schema R
  - Relation s has schema S

$R \cap S$
- The set of attributes that R and S have in common

$R - S$
- The set of attributes in R that are not also in S
- (and, the attributes are in the same order as R)

$K \subseteq R$
- K is some subset of the attributes in relation schema R

Attribute Domains

- The relational model constrains attribute domains to be *atomic*
  - Values are indivisible units
- Mainly a simplification
  - Virtually all relational database systems provide non-atomic data types
- Attribute domains may also include the *null* value
  - *null* = the value is unknown or unspecified
  - *null* can often complicate things. Generally considered good practice to avoid wherever reasonable to do so.

Relations and Relation Variables

More formally:

- *account* is a relation variable
  - A name associated with a specific schema, and a set of tuples that satisfies that schema
  - (sometimes abbreviated "relvar")
- A specific set of tuples with the same schema is called a relation value (sometimes abbreviated "relval")
  - (Formally, this can also be called a relation)
  - Can be associated with a relation variable
  - Or, can be generated by applying relational operations to one or more relation variables

acct_id	branch_name	balance
A-301	New York	350
A-307	Seattle	275
A-318	Los Angeles	550
. . .	. . .	. . .

account

Relations and Relation Variables (2)

- Problem:
  - The term "relation" is often used in slightly different ways
- "Relation" usually means the collection of tuples
  - i.e. "relation" usually means "relation value"
- It is often used less formally to refer to a relation variable and its associated relation value
  - e.g. "the account relation" is really a relation variable that holds a specific relation value

acct_id	branch_name	balance
A-301	New York	350
A-307	Seattle	275
A-318	Los Angeles	550
. . .	. . .	. . .

account

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

Distinguishing Tuples

- Relations are sets of tuples…
  - No two tuples can have the same values for *all* attributes…
  - But, some tuples might have the same values for some attributes
- Example:
  - Some accounts have the same balance
  - Some accounts are at the same branch

acct_id	branch_name	balance
A-301	New York	350
A-307	Seattle	275
A-318	Los Angeles	550
A-319	New York	80
A-322	Los Angeles	275

account

Keys

- Keys are used to distinguish individual tuples
  - A superkey is a set of attributes that uniquely identifies tuples in a relation
- Example:

  {*acct_id*} is a superkey
- Is {*acct_id, balance*} a superkey?
  - Yes! Every tuple will have a unique set of values for this combination of attributes.
- Is {*branch_name*} a superkey?
  - No. Each branch can have multiple accounts

acct_id	branch_name	balance
A-301	New York	350
A-307	Seattle	275
A-318	Los Angeles	550
A-319	New York	80
A-322	Los Angeles	275

account

Superkeys and Candidate Keys

- A superkey is a set of attributes that uniquely identifies tuples in a relation
- Adding attributes to a superkey produces another superkey
  - If {*acct_id*} is a superkey, so is {*acct_id*, *balance*}
  - If a set of attributes $K \subseteq R$ is a superkey, so is any superset of K
  - Not all superkeys are equally useful…
- A *minimal* superkey is called a candidate key
  - A superkey for which no proper subset is a superkey
  - For account, only {*acct_id*} is a candidate key

Primary Keys

- A relation might have several candidate keys
- In these cases, one candidate key is chosen as the primary means of uniquely identifying tuples
  - Called a primary key
- Example: customer relation
  - Candidate keys could be:
    {*cust_id*}
    {*cust_ssn*}
  - Choose primary key:
    {*cust_id*}

cust_id	branch_name	balance
23-652	Joe Smith	330-25-8822
15-202	Ellen Jones	221-30-6551
23-521	Dave Johnson	005-81-2568
. . .	. . .	. . .

customer

Primary Keys (2)

- Keys are a property of the relation schema, not individual tuples
  - Applies to *all tuples* in the relation
- Primary key attributes are listed first in relation schema, and are underlined
- Examples:

  Account_schema = (*acct_id*, *branch_name*, *balance*)

  Customer_schema = (*cust_id*, *cust_name*, *cust_ssn*)
- Only indicate primary keys in this notation
  - Other candidate keys are not specified

Primary Keys (3)

- Multiple records cannot have the same values for a primary key!
  - …or any candidate key, for that matter…
- Example:

  customer(*cust_id*, *cust_name*, *cust_ssn*)
  - Two customers cannot have the same ID.
- This is an example of an invalid relation
  - The set of tuples doesn't satisfy the required constraints

cust_id	branch_name	balance
23-652	Joe Smith	330-25-8822
15-202	Ellen Jones	221-30-6551
23-521	Dave Johnson	005-81-2568
15-202	Albert Stevens	450-22-5869
. . .	. . .	. . .

customer

Keys Constrain Relations

- Primary keys constrain the set of tuples that can appear in a relation
  - Same is true for *all superkeys*
- For a relation r with schema R
  - If $K \subseteq R$ is a superkey then

    $\langle \forall t_1, t_2 \in r(R) : t_1[K] = t_2[K] : t_1[R] = t_2[R] \rangle$
  - i.e., if two tuple-variables have the same values for the superkey attributes, then they refer to the same tuple
    - $t_1[R] = t_2[R]$ is equivalent to saying $t_1 = t_2$

Choosing Candidate Keys

- Since candidate keys constrain the tuples that can be stored in a relation…
  - Attributes that would make good (or bad) candidate keys depend on *what is being modeled*
- Example: customer name as candidate key?
  - Very likely that multiple people will have same name
  - Thus, not a good idea to use it as a candidate key
- These constraints motivated by external requirements
  - Need to understand what we are modeling in the database

Super Key vs. Candidate Key vs. Primary Key

Super Key

- *Definition*: A super key is an attribute or a set of attributes that can uniquely identify a record in a database table. A super key may contain additional attributes that are not necessary for unique identification.
- *Uniqueness*: Super keys guarantee that no two rows have the same set of values in the super key columns. However, they are not minimal, meaning a super key can have extra attributes that aren't needed to maintain uniqueness.
- *Multiplicity*: A table can have multiple super keys, varying in the number of attributes they include, as long as the combination ensures uniqueness.

Super Key vs. Candidate Key vs. Primary Key

Candidate Key

- *Definition*: A candidate key is a minimal super key, meaning it is a super key with no possible reduction in the number of columns that can still uniquely identify a record. Essentially, it is the minimal subset of a super key.
- *Uniqueness and Non-nullability*: Each candidate key must contain unique values and cannot contain null values, ensuring that every record within the table can be uniquely identified by this key.
- *Multiplicity*: A table can have multiple candidate keys if there are multiple minimal sets of fields that can uniquely identify a record in the table.

Super Key vs. Candidate Key vs. Primary Key

Primary Key

- *Definition*: The primary key is a specific candidate key selected by the database designer to uniquely identify records in a table. It is the most important key and is used to reference other tables (foreign keys).
- *Uniqueness and Non-nullability*: The primary key must consist of unique values and cannot contain null values. It enforces entity integrity by uniquely identifying each record in a table.
- *Singular Presence*: A table can have only one primary key. This can be a single column (simple primary key) or composed of multiple columns (composite primary key) to ensure uniqueness across records.

Foreign Keys

- One relation schema can include the attributes of another schema's primary key
- Example: depositor relation
  - *Depositor_schema* = (*cust_id*, *acct_id*)
  - Associates customers with bank accounts
  - *cust_id* and *acct_id* are both foreign keys
    - *cust_id* references the primary key of *customer*
    - *acct_id* references the primary key of *account*
  - depositor is the referencing relation
    - It refers to the customer and account relations
  - customer and account are the referenced relations

depositor Relation

cust_id	cust_name	cust_ssn
23-652	Joe Smith	330-25-8822
15-202	Ellen Jones	221-30-6551
23-521	Dave Johnson	005-81-2568
. . .	. . .	. . .

customer

acct_id	branch_name	balance
A-301	New York	350
A-307	Seattle	275
A-318	Los Angeles	550
. . .	. . .	. . .

account

cust_id	acct_id
15-202	A-301
23-521	A-307
23-652	A-318
. . .	. . .

depositor

- *depositor* relation references *customer* and *account*
- Represents relationships between customers and their accounts
- Example: Joe Smith's accounts
  - "Joe Smith" has an account at the "Los Angeles" branch, with a balance of 550.

Foreign Key Constraints

- Tuples in *depositor* relation specify values for *cust_id*
  - *customer* relation must contain a tuple corresponding to each *cust_id* value in *depositor*
- Same is true for *acct_id* values and *account* relation
- Valid tuples in a relation are also constrained by foreign key references
  - Called a foreign-key constraint
- Consistency between two dependent relations is called referential integrity
  - Every foreign key value must have a corresponding primary key value

Foreign Key Constraints (2)

- Given a relation $r(R)$
  - A set of attributes $K \subseteq R$ is the primary key for R
- Another relation $s(S)$ references r
  - $K \subseteq S$ too
  - $\langle \forall t_s \in s : \exists t_r \in r : t_s[K] = t_r[K] \rangle$
- Notes:
  - K is not required to be a candidate key for S, only R
  - K may also be part of a larger candidate key for S

Primary Key of depositor Relation?

- *Depositor_schema* = (*cust_id*, *acct_id*)
- If {*cust_id*} is the primary key:
  - A customer can only have one account
    - Each customer's ID can appear only once in *depositor*
  - An account could be owned by multiple customers
- If {*acct_id*} is the primary key:
  - Each account can be owned by only one customer
    - Each account ID can appear only once in *depositor*
  - Customers could own multiple accounts
- If {*cust_id*, *acct_id*} is the primary key:
  - Customers can own multiple accounts
  - Accounts can be owned by multiple customers
- Last option is how most banks really work

cust_id	acct_id
15-202	A-301
23-521	A-307
23-652	A-318
. . .	. . .

depositor

Conceptual Data Model (CDM)

- Model the logical structure of the entire data application, independent of software or data structure model considerations
- A valid CDM can be converted into a PDM
- In its application, CDM can be equated with ERD
  - whose function is indeed the same, namely modeling the logical structure of the database
- CDM is used to describe in detail the logical structure of the database
- CDM consists of objects that are not directly implemented into the actual database

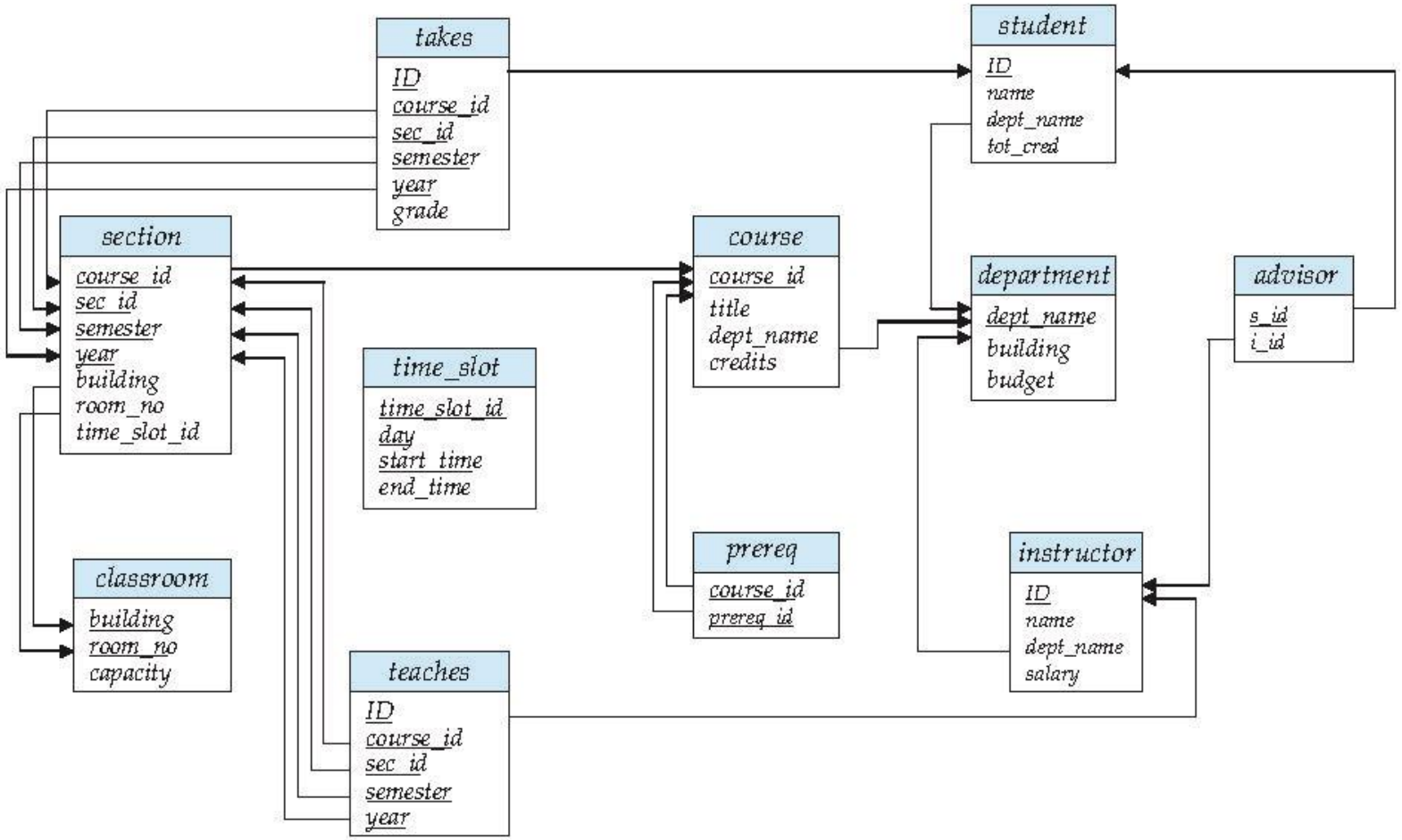
Steps to Create a CDM

- Understand the core of the problem given
- Determine which entities are involved
- Determine the data attributes for each entity along with their data types
- Identify the relationships/associations between each entity including their cardinalities
- Model the Entities and Relationships
- Verify the accuracy of the model
- Correct any errors and warnings

Physical Data Model (PDM)

- A physical representation of the database to be created, taking into account the DBMS to be used
- PDM can be generated from a valid CDM
- In its application, PDM can be equated with the Relation Schema whose function is to model the physical structure of a database
- It is a detailed representation of a database in physical form
- PDM shows the correct data storage structure in the actual database used

Schema Diagram for University Database



Relational Query Languages

- Procedural vs non-procedural, or declarative
- "Pure" languages:
  - Relational algebra
  - Tuple relational calculus
  - Domain relational calculus
- The above 3 pure languages are equivalent in computing power
- We will concentrate in this chapter on relational algebra
  - Not turing-machine equivalent
  - consists of 6 basic operations

Select Operation – selection of rows (tuples)

- 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

Project Operation – Selection of Columns (Attributes)

- Relation r

A	B	C
α	10	1
α	20	1
β	30	1
β	40	2

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

A	C
α	1
α	1
β	1
β	2

=

A	C
α	1
β	1
β	2

Union of Two Relations

- Relations *r, s*

A	B
α	1
α	2
β	1

r

A	B
α	2
β	3

s

- $r \cup s$

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

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

Cartesian-product – Naming Issue

- 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

Renaming a Table

- Allows us to refer to a relation, (say E) by more than one name

$$\rho_x(E)$$

returns the expression E under the name X

- Relations r

A	B
α	1
β	2

r

- $r \times \rho_s(r)$

$r.A$	$r.B$	$s.A$	$s.B$
α	1	α	1
α	1	β	2
β	2	α	1
β	2	β	2

Composition of Operations

- Can build expressions using multiple operations

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

  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

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

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

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

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

Notes about Relational Languages

- Each Query input is a table (or set of tables)
- Each query output is a table.
- All data in the output table appears in one of the input tables
- Relational Algebra is not Turning complete
- Can we compute:
  - SUM
  - AVG
  - MAX
  - MIN

Summary of Relational Algebra Operators

Symbol (Name)	Example of Use
σ (Selection)	$\sigma_{salary >= 85000}(instructor)$
	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.
x (Cartesian Product)	*instructor* x *department*
	Output pairs of rows from the two input relations that have the same value on all attributes that have the same name.
∪ (Union)	$\Pi_{name}(instructor) \cup \Pi_{name}(student)$
	Output the union of tuples from the *two* input relations.
– (Set Difference)	$\Pi_{name}(instructor) - \Pi_{name}(student)$
	Output the set difference of tuples from the two input relations.
⋈ (Natural Join)	*instructor* ⋈ *department*
	Output pairs of rows from the two input relations that have the same value on all attributes that have the same name.

Thank you!