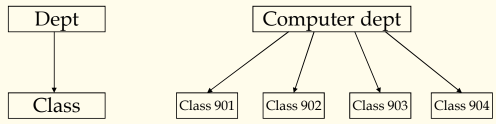
2. Data Model*

2.1 Hierarchical Data Model

Basic idea: because many things in real world are organized in hierarchy, hierarchical model manages to describe real world in a tree structure.

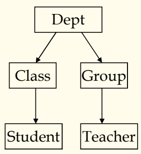
- Record and field
- Parent-Child relationship (PCR): the most basic data relationship in hierarchical model. It expresses a 1:N relationship between two record types.



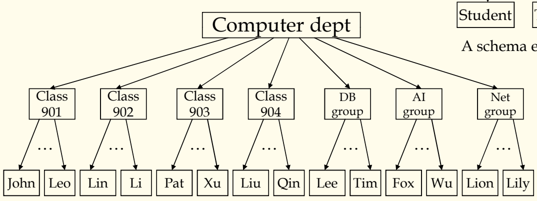
A PCR type   An instance of PCR

Hierarchical Data Schema

- A hierarchical data schema consists of PCRs.
- Every PCR expresses one 1:N relationship
- Every record type can only have one parent



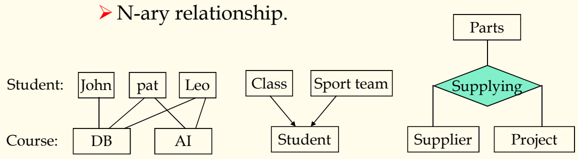
A schema example



An instance of hierarchical data schema

Virtual Record

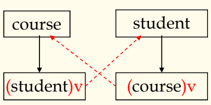
- In real world, many data are not hierarchical. It is hard to express them directly with PCR.
  - M:N relationship between different record types
  - A record type is the child of more than two PCRs.
  - N-ary relationship.



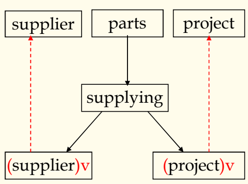
A M:N relationship   Multi parents   Ternary relationship

Virtual Record

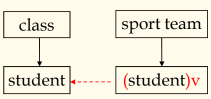
- To avoid redundant, virtual record is introduced to express above relationships. It is a pointer in fact.



M:N expressed with virtual record type



Ternary relationship expressed with virtual record type

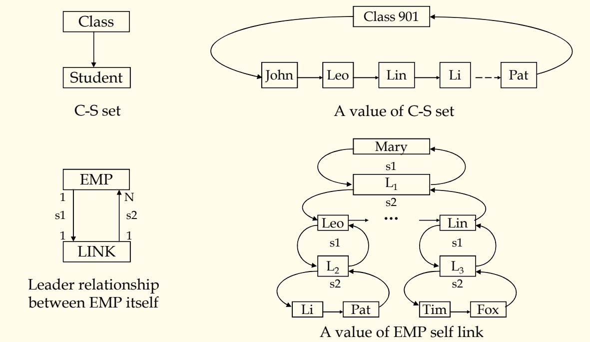


Multi parent expressed with virtual record type

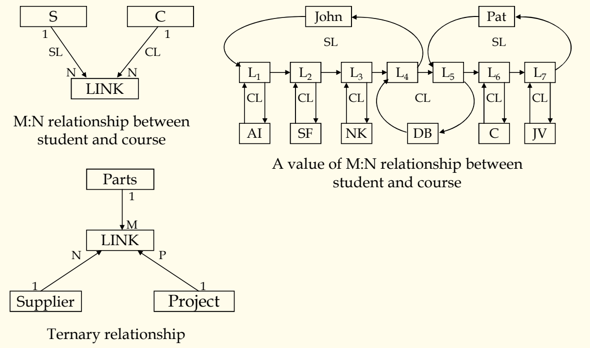
2.2 Network Data Model

- The basic data structure is "set", it represent a 1:N relationship between things in real world. "1" side is called owner, and "N" side is called member. One record type can be the owner of multi sets, and also can be the member of multi sets. Many sets form a network structure to express real world.
- It breakthrough the limit of hierarchical structure, so can express non-hierarchical data more easy.
- Record and data items: data items are similar as field in hierarchical model, but it can be vector.
- Set : express the 1:N relationship between two record types.
- LINK record type: used to express self relationship, M:N relationship and N-ary relationship.

Example of Network Data Schema



Example of Network Data Schema



2.3 Relational Data Model

- The basic data structure is "table", or relation. The things and the relationships between them in real world are all expressed as tables, so it can be researched in strict mathematic methods. It raises the database technology to a theory height. Its features:
  - ✓ Based on set theory, high abstract level
  - ✓ Shield all lower details, simple and clear, easy to understand
  - ✓ Can establish new algebra system——relational algebra
  - ✓ Non procedure query language——SQL
  - ✓ *Soft link* ——the essential difference with former data models

Understand *Soft link*

student

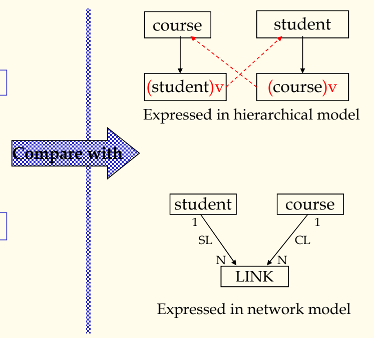
S#	SNAME	AGE	…

elective

S#	*C#*	*GRADE*

course

C#	CNAME	SCR	…



Attributes and Domain

- The features of an entity in real world are expressed as attributes in relational model

  E.g. a student can be described with the attributes such as *name, sid, gender, age, birthday, nationality*, etc.
- The value scope of an attribute is called its domain.
  - ➢ Atomic data --- 1NF
  - ➢ Null

Relation and Tuple

- An entity of real world can be expressed as one or more than one relations.
- A relation is a N-ary relationship defined on all of its attribute domain.

  Suppose a relation R with attributes $A_1, A_2, \dots A_n$, the corresponding domains are $D_1, D_2, \dots D_n$, then R can be expressed as:

  $R = (A_1/D_1, A_2/D_2, \dots A_n/D_n)$, or

  $R = (A_1, A_2, \dots A_n)$
- This is called the schema of R, and n is the number of attributes, called the degree of R. $A_i (1 \le i \le n)$ is attribute name.

Relation and Tuple

- An instance (value) of R can be expressed as r or r(R), it is a set of n-tuple:
  $r = \{t_1, t_2, \ldots, t_m\}$
  every tuple t can be expressed as:
  $t = <v_1, v_2, \ldots, v_n>, v_i \in D_i, 1 \le i \le n$
  that is:
  $t \in D_1 \times D_2 \times, \ldots, \times D_n, 1 \le i \le n$
  that is:
  $r \subseteq D_1 \times D_2 \times, \ldots, \times D_n, 1 \le i \le n$
- Relation is also called *table*. Attribute is also called *column*, and tuple is also called *row*.

Primary Key

- A set of attributes is a *candidate key* for a relation if :
  1. No two distinct tuples can have same values in this set of attributes, and
  2. This is not true for any subset of this set of attributes.
  - Part 2 false? A *superkey*.
  - If there's >1 key for a relation, one of the keys is chosen (by DBA) to be the *primary key*, and the others are called *alternate key*.
  - If the *primary key* consists of all attributes of a relation, it is called *all key*.
- That means, the key can decide a tuple uniquely.
- E.g., *sid* is a key for Students. (What about *name*?) The set {*sid*, *gpa*} is a superkey.

Foreign Keys, Referential Integrity

- *Foreign key* : Set of attributes in one relation that is used to 'refer' to a tuple in another relation. (Must correspond to primary key of the second relation.) Like a 'logical pointer'.
- E.g. *sid* is a foreign key referring to Students:
  - Enrolled(*sid*: string, *cid*: string, *grade*: string)
  - If all foreign key constraints are enforced, *referential integrity* is achieved, i.e., no dangling references.
  - Have you forgotten *soft link*?

An Example of Referential Integrity

- Only students listed in the Students relation should be allowed to enroll for courses.

Enrolled

sid	cid	grade
53666	Carnatic101	C
53666	Reggae203	B
53650	Topology112	A
53666	History105	B

Students

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

Other Integrity Constraints

- Domain integrity constraint
  - An attribute's value must be a value in the domain of this attribute. This is the most basic constraint. All popular RDBMS are able to check domain integrity constraint automatically.
- Entity integrity constraint
  - Every relation should have a primary key. The value of primary key of each tuple must be unique. Primary key cannot be *NULL*. This is so-called entity integrity constraint.

Example Instances

- "Sailors", "Reserves" and "Boats" relations for our examples.

R1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

B1

bid	bname	color
101	tiger	red
103	lion	green
105	hero	blue

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

S2

sid	sname	rating	age
28	yuppy	9	35.0
31	lubber	8	55.5
44	guppy	5	35.0
58	rusty	10	35.0

Relational Algebra

- Basic operations:
  - *Selection* (σ) Selects a subset of rows from relation.
  - *Projection* (π) Deletes unwanted columns from relation.
  - *Cross-product* (×) Allows us to combine two relations.
  - *Set-difference* (−) Tuples in reln. 1, but not in reln. 2.
  - *Union* (∪) Tuples in reln. 1 and in reln. 2.
- {σ, π, ∪, −, ×} is a complete operation set. Any other relational algebra operations can be derived from them.
- Additional operations:
  - Intersection, ***join***, division, outer join, outer union: Not essential, but (very!) useful.
- Since each operation returns a relation, operations can be composed! (Algebra is "closed".)

Projection

- Deletes attributes that are not in *projection list*.
- *Schema* of result contains exactly the fields in the projection list, with the same names that they had in the input relation.
- Projection operator has to eliminate *duplicates*! (Why??)
  - Note: real systems typically don't do duplicate elimination unless the user explicitly asks for it. (Why not?)

sname	rating
yuppy	9
lubber	8
guppy	5
rusty	10

$\pi_{sname,rating}(S2)$

age
35.0
55.5

$\pi_{age}(S2)$

Selection

- Selects rows that satisfy *selection condition*.
- No duplicates in result! (Why?)
- *Schema* of result identical to that of input relation.
- *Result* relation can be the *input* for another relational algebra operation! (*Operator composition.*)

sid	sname	rating	age
28	yuppy	9	35.0
58	rusty	10	35.0

$\sigma_{rating>8}(S2)$

sname	rating
yuppy	9
rusty	10

$\pi_{sname,rating}(\sigma_{rating>8}(S2))$

Union, Intersection, Set-Difference

- All of these operations take two input relations, which must be *union-compatible*:
  - Same number of fields.
  - Corresponding attributes have the same type.
- What is the *schema* of result?

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0
44	guppy	5	35.0
28	yuppy	9	35.0

$S1 \cup S2$

sid	sname	rating	age
22	dustin	7	45.0

$S1 - S2$

sid	sname	rating	age
31	lubber	8	55.5
58	rusty	10	35.0

$S1 \cap S2$

Cross-Product

- Each row of S1 is paired with each row of R1.
- *Result schema* has one attribute per attribute of S1 and R1, with attribute names *inherited* if possible.
  - *Conflict*: Both S1 and R1 have an attribute called *sid*.

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	22	101	10/10/96
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	22	101	10/10/96
31	lubber	8	55.5	58	103	11/12/96
58	rusty	10	35.0	22	101	10/10/96
58	rusty	10	35.0	58	103	11/12/96

- *Renaming operator:* $\rho(C(1 \to sid1, 5 \to sid2), S1 \times R1)$

Joins

- *Condition Join* : $R \bowtie_C S = \sigma_C (R \times S)$

(sid)	sname	rating	age	(sid)	bid	day
22	dustin	7	45.0	58	103	11/12/96
31	lubber	8	55.5	58	103	11/12/96

$S1 \bowtie_{S1.sid<R1.sid} R1$

- *Result schema* same as that of cross-product.
- Fewer tuples than cross-product, might be able to compute more efficiently
- Sometimes called a *theta-join*.

Joins

- *Equi-Join*: A special case of condition join where the condition c contains only ***equalities.***

sid	sname	rating	age	bid	day
22	dustin	7	45.0	101	10/10/96
58	rusty	10	35.0	103	11/12/96

$$S1 \bowtie_{sid} R1$$

- *Result schema* similar to cross-product, but only one copy of attributes for which equality is specified.
- *Natural Join*: Equi-join on *all* common attributes.

Division

- Not supported as a primitive operator, but useful for expressing queries like:
  - *Find sailors who have reserved **all** boats.*
- Let A have 2 fields, x and y; B have only field y:
  - $A/B = \{\langle x\rangle \mid \exists \langle x, y\rangle \in A \ \forall \langle y\rangle \in B\}$
  - i.e., **A/B contains all x tuples (sailors) such that for *every* y tuple (boat) in B, there is an xy tuple in A.**
  - *Or*: If the set of y values (boats) associated with an x value (sailor) in A contains all y values in B, the x value is in A/B.
- In general, x and y can be any lists of fields; y is the list of fields in B, and $x \cup y$ is the list of fields of A.

Examples of Division A/B

A

sno	pno
s1	p1
s1	p2
s1	p3
s1	p4
s2	p1
s2	p2
s3	p2
s4	p2
s4	p4

B1

pno
p2

B2

pno
p2
p4

B3

pno
p1
p2
p4

A/B1

sno
s1
s2
s3
s4

A/B2

sno
s1
s4

A/B3

sno
s1

Expressing A/B Using Basic Operators

- Division is not essential op; just a useful shorthand.
  - (Also true of joins, but joins are so common that systems implement joins specially.)
- *Idea*: For A/B, compute all x values that are not *disqualified* by some y value in B.
  - x value is *disqualified* if by attaching y value from B, we obtain an xy tuple that is not in A.

Disqualified x values: $\pi_x((\pi_x(A) \times B) - A)$

A/B: $\pi_x(A)$ − all disqualified tuples

Outer Joins

- The extension of join operation. In join operation, only matching tuples fulfilling join conditions are left in results. Outer joins will keep unmated tuples, the vacant part is set *Null* :
  - Left outer join(*⋈)
    Keep all tuples of left relation in the result.
  - Right outer join (⋈*)
    Keep all tuples of right relation in the result.
  - Full outer join (*⋈*)
    Keep all tuples of left and right relations in the result.

Examples of Outer Joins

S1

sid	sname	rating	age
22	dustin	7	45.0
31	lubber	8	55.5
58	rusty	10	35.0

R1

sid	bid	day
22	101	10/10/96
58	103	11/12/96

S1 *⋈ R1

sid	sname	rating	age	bid	day
22	dustin	7	45.0	101	10/10/96
58	rusty	10	35.0	103	11/12/96
31	Lubber	8	55.5	null	null

S1⋈* R1 = S1⋈R1 **(Why?)**

sid	sname	rating	age	bid	day
22	dustin	7	45.0	101	10/10/96
58	rusty	10	35.0	103	11/12/96

S1 *⋈* R1

sid	sname	rating	age	bid	day
22	dustin	7	45.0	101	10/10/96
58	rusty	10	35.0	103	11/12/96
31	Lubber	8	55.5	null	null

Outer Unions

- The extension of union operation. It can union two relations which are not *union-compatible*.
- The attribute set in result is the union of attribute sets of two operands.
- The values of attributes which don't exist in original tuples are filled as NULL

sid	sname	rating	age	bid	day
22	dustin	7	45.0	null	null
31	Lubber	8	55.5	null	null
58	rusty	10	35.0	null	null
22	null	null	null	101	10/10/96
58	null	null	null	103	11/12/96

S1∪**R1**

Relational Calculus

- Relational Algebra needs to specify the order of operations; while relational calculus only needs to indicate the logic condition the result must be fulfilled.
- Comes in two flavors: *Tuple relational calculus* (TRC) and *Domain relational calculus* (DRC).
- Calculus has *variables, constants, comparison ops, logical connectives* and *quantifiers*.
  - *TRC*: Variables range over (i.e., get bound to) *tuples*.
  - *DRC*: Variables range over *domain elements* (attribute values).
  - Both TRC and DRC are simple subsets of first-order logic.
- Expressions in the calculus are called *formulas*. An answer tuple is essentially an assignment of constants to variables that make the formula evaluate to *true*.

Domain Relational Calculus

- *Query* has the form:

  $\{ <x_1,x_2, \ldots, x_n> \mid P(x_1,x_2,\ldots,x_n,x_{n+1},\ldots, x_{n+m}) \}$
- $x_1,x_2,\ldots,x_n,x_{n+1},\ldots, x_{n+m}$ are called *domain variables*. $x_1,x_2,\ldots,x_n$ appear in result.
- *Answer* includes all tuples $<x_1,x_2, \ldots, x_n>$ that make the *formula* $P(x_1,x_2,\ldots,x_n,x_{n+1},\ldots, x_{n+m})$ be *true*.
- *Formula* is recursively defined, starting with simple *atomic formulas* (getting tuples from relations or making comparisons of values), and building bigger and better formulas using the *logical connectives*.

DRC Formulas

- *Atomic formula:*
  - $<x_1,x_2, \ldots, x_n> \in Rname$, or X *op* Y, or X *op* constant
  - *op* is one of $<,>,=,\leq,\geq,\neq$
- *Formula:*
  - an atomic formula, or
  - $\neg p$, $p \wedge q$, $p \vee q$, where p and q are formulas, or
  - $\exists X(p(X))$, where variable X is *free* in p(X), or
  - $\forall X(p(X))$, where variable X is *free* in p(X)

Free and Bound Variables

- The use of quantifiers $\exists X$ and $\forall X$ is said to *bind* X.
  - A variable that is not bound is free.
- Let us revisit the definition of a query:

  $<x_1,x_2, \ldots, x_n> \mid P(x_1,x_2,\ldots,x_n,x_{n+1},\ldots, x_{n+m})$
- There is an important restriction: the variables $x_1,x_2, \ldots, x_n$ that appear to the left of '|' must be the *only* free variables in the formula p(...).

Find all sailors with a rating above 7

- $\{ <I,N,T,A> \mid <I,N,T,A> \in Sailors \wedge T>7 \}$
- The condition $<I,N,T,A> \in Sailors$ ensures that the domain variables *I, N, T* and *A* are bound to fields of the same Sailors tuple.
- The term $<I,N,T,A>$ to the left of '|' (which should be read as *such that*) says that every tuple $<I,N,T,A>$ that satisfies $T>7$ is in the answer.
- Modify this query to answer:
  - Find sailors who are older than 18 or have a rating under 9, and are called 'Joe'.

Unsafe Queries, Expressive Power

- It is possible to write syntactically correct calculus queries that have an infinite number of answers! Such queries are called *unsafe*.
  - e.g., $\{S \mid \neg(S \in Sailors)\}$
- It is known that every query that can be expressed in relational algebra can be expressed as a safe query in DRC / TRC; the converse is also true.
- *Relational Completeness:* $\{\sigma, \pi, \cup, -, \times\}$ is a complete operation set. Relational calculus can express these five operations easily, so relational calculus is also *Relational Completeness*. SQL language is based on relational calculus, so it can express any query that is expressible in relational algebra /calculus.

Tuple Relational Calculus

- *Query* has the form:
  { t[<*attribute list*>] | P(t)}
- t is called *tuple variable.*
- ✓ *Answer* includes all tuples t<*attribute list*> that make the *formula* P(t) be *true*.
- ✓ Example query: Find all sailors' name whose rating above 7 and younger than 50;
  $\{ t[N] \mid t \in Sailors \wedge t.T>7 \wedge t.A<50 \}$

Remarks to Traditional Data Model

- Hierarchical, Network, and Relational Model
- Suitable for OLTP applications
- Based on record, can't orient to users or applications better
- Can't express the relationships between entities in a natural mode.
- Lack of semantic information
- Few data type, hard to fulfill the requirements of applications

2.4 ER Data Model

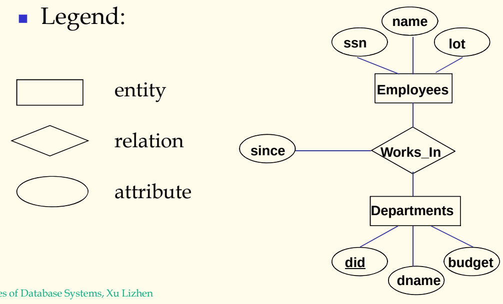
- *Entity:* Real-world object distinguishable from other objects. An entity is described (in DB) using a set of *attributes*.
- *Entity Set:* A collection of similar entities. E.g., all employees.
  - All entities in an entity set have the same set of attributes. (Until we consider ISA hierarchies, anyway!)
  - Each entity set has a *key*.
  - Each attribute has a *domain*.
  - Permit combined or multi-valued attribute

Relationship

- *Relationship:* Association among two or more entities. E.g., Attishoo *works in* Pharmacy department.
  - Relationship can have attributes
- *Relationship Set:* Collection of similar relationships.
  - An n-ary relationship set R relates n entity sets E_1 ... E_n; each relationship in R involves entities $e_1, \ldots, e_n$
  - Same entity set could participate in different relationship sets, or in different "roles" in same set.

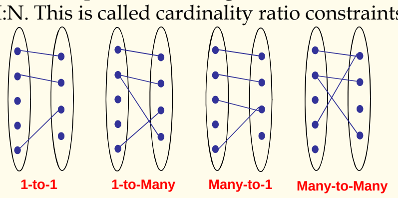
ER Diagram

- Concept model: entity—relationship, be independent of practical DBMS.
- Legend:



Cardinality Ratio Constraints

- Relationships can be distinguished as 1:1, 1:N, and M:N. This is called cardinality ratio constraints.



- For example: an employee can work in many departments; a dept can have many employees. This M:N. In contrast, each dept has at most one manager and one employee can only be manager of one dept, then this is 1:1.

Participation Constraints

- We can further specify the minimal and max number an entity participates a relationship. This is called participation constraints.
- If a department must have a manager, then we say the Departments is *total participation* in Manages relationship (*vs. partial*). The minimal participating degree of Departments is 1.
- Another example: in the selected course relationship between Students and Courses, if we specify every student must select at least 3 courses and at most 6 courses, the participating degree of Students is said to be (3,6).

Advanced Topics of ER Model

- Weak entity
- Specialization and Generalization
  - Similar as inheriting in Object-Oriented data model
- Aggregation
  - Allows us to treat a relationship set as an entity set for purposes of participation in (other) relationships
- Category
  - Allow us to express an entity set consists of different types of entities. That is, a hybrid entity set.

2.5 Object-Oriented Data Model

- The shortage of relational data model
- Break through 1NF
- Object-Oriented analysis and programming
- Requirement of objects' permanent store
- Object-Relation DBMS
- Native (pure) Object-Oriented DBMS

2.6 Other Data Models

- Logic-based data model (Deductive DBMS)
  - Extend the query function of DBMS (especially recursive query function)
  - Promote the deductive ability of DBMS
- Temporal data model
- Spatial data model
- XML data model
  - Store data on internet
  - Common data exchange standard
  - Information systems integration
  - Expression of semi-structured data
  - … …
- Others

2.7 Summary

- Data model is the core of a DBMS
- A data model is a methodology to simulate real world in database
- In fact, every kind of DBMS has implemented a data model

☹ If there will be a data model which can substitute relational model and become popular data model, just as relational model substituted hierarchical and network model 30 years ago **???**