Design Principles

Setting: client has (possibly vague) idea of what is required. You must design a database that represents these thoughts and only these thoughts.

Important note:

Avoid redundancy,i.e., saying the same thing more than once. It wastes space and encourages inconsistency.

Example

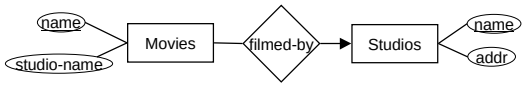
Good:

1

Example

This is a bad design…

That's also a bad design…



2

Use Schema to Enforce Constraints

The design *schema* should enforce as many constraints as possible.

- Don't assume future data will follow assumptions.

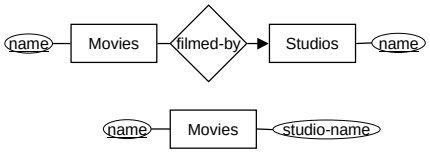
Example:

- If a user must associate only one director with a movie, don't allow sets of directors and count on users to enter only one director per movie.

3

Entity Sets Vs. Attributes

- You may be unsure which concepts are worthy of being entity sets, and which are handled more simply as attributes.



4

Intuitive Rule for E.S. Vs. Attribute

Make an entity set only if either:

- Is more than the 'name' of something; *i.e.*, it has non-key attributes or it has relationships with a number of different entity sets, or
- Is the "many" in a many-one relationship.

5

Example

The following design illustrates both points:

- *Studios* deserves to be an E.S. because we record *addr*, a non-key attribute.
- *Movies* deserves to be an E.S. because it is at the "many" end.

6

Don't Overuse Weak E.S.

- There is a tendency to feel that no E.S. has its entities uniquely determined without following some relationships.
- However, in practice, we almost always create unique ID's to compensate: social-security numbers, OHIP's, etc.
- The only times weak E.S.'s seem necessary are when:
  - We can't easily create such ID's; e.g., no one is going to accept a "species ID" as part of the standard nomenclature (species is a weak E.S. supported by membership in a genus).
  - There is no global authority to create them, *e.g.*, crews and studios.

7

The Relational Model

8

Relational Model

- Table = relation.
- Column headers = attributes.
- Attribute domains (atomic)
- Row = tuple

Relation stars-in

Movies	Stars
Basic Instinct	Sharon Stone
Total Recall	Arnold Schwarzenegger
Total Recall	Sharon Stone
…	…

- Relation schema = name(attributes) + other structure info., e.g., keys, other constraints.
  Example: *stars-in*(*movie-name*, *star-name*)
  - Order of attributes is arbitrary, but in practice we need to assume the order given in the relation schema.
- Relation instance is current set of rows for a relation schema.
- Database schema is a collection of relation schemas.

9

Why Relations?

- Very simple model.
- *Often* a good match for the way we think about our data.
- Data base query languages (e.g., SQL) manipulate such relational tables..

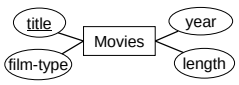
10

Relational Design

We start with an E/R design. Then, the simplest approach (but not always best) is :

- convert each E.S. to a relation and
- convert each relationship to a relation.

Example:



Movies(*title*, *year*, *length*, *film-type*)

Note: E.S. attributes become relational attributes

11

Keys in Relations

An attribute (or set of attributes) *K* is a *key* for a relation *R* if we expect that in no instance of *R* will two different tuples agree on all the attributes of *K*. We indicate a key by underlining the key attributes. (pretty much what we did in E/R)

Example:

If *title* is a key for *Movies, then*
Movies(*title*, *year*, *length*, *film-type*)

12

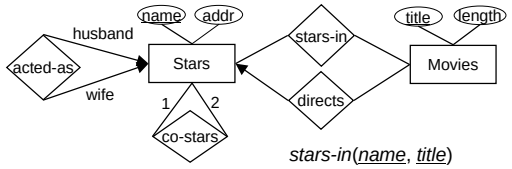
From E/R Relationships to Relations

The constructed relation has the following attributes:

- The *key* attributes of each E.S. that participates in the relationship.
- Any attributes that belong to the relationship itself.
- Renaming attributes is necessary if an E.S. has multiple roles in the relationship

13

From E/R Relationships to Relations (cont.)



stars-in(*name*, *title*)
directs(*title*, *name*)
*acted-as**(*husband*, *wife*)
co-stars(*name1*, *name2*)

(*) For *acted-as*, we can choose either *husband* or *wife* as key

14

Combining Relations

Common case: Combine relation for an E.S. *E* with the relation for some many-one relationship from *E* to another E.S. *F*

Example: Combine *Movies*(*title*, length) with *directs*(*title*, name) to get *Movies-new*(*title*, *length*, *director-name*).

However danger in pushing this idea too far: redundancy

Example: Combine *Movies*(*title*, length) with *stars-in*(*name*, *title*) to get *Movies-stars*(*title*, *name*, *length*,)

title	name	length
Star Wars II	Ewan McGregor	142
Star Wars II	Natalie Portman	142

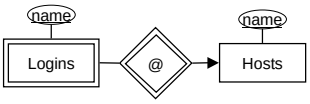
15

Weak Entity Sets & Relationships to Relations

- Relation for a weak E.S. must include its full key (*i.e.*, attributes of related entity sets) as well as its own attributes.
- A supporting (double-diamond) relationship yields a relation that is actually redundant and should be deleted from the database schema.

16

Weak Entity Sets & Relationships to Relations (cond.)



Hosts(*name*)
Logins(*name*, *hname*)
At(*lname*, *hname*, *hname2*)

- In *At*, *hname* and *hname2* must be the same host, so delete one of them.
- Then, *Logins* and *At* become the same relation; delete one of them.
- In this case, *Hosts*' schema is a subset of Logins' schema. Delete *Hosts*?

17

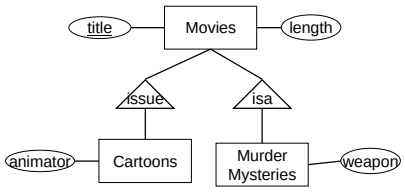
Subclasses to Relations

Three approaches:

1. Object-oriented: each entity is in one class. Create a relation for each class, with all the attributes for that class.
   - Don't forget inherited attributes.
2. E/R style: an entity is in a network of classes related by *isa*. Create one relation for each E.S.
   - An entity is represented in the relation for each subclass to which it belongs.
   - Relation has only the attributes attached to that E.S. + key.
3. Use nulls. Create one relation for the root class or root E.S., with all attributes found anywhere in its network of subclasses.
   - Put `NULL` in attributes not relevant to a given entity.

18

Example



Subclasses to Relations: An Example

OO Style:

Title	*Length*	*Weapon*
Episode I	133	Lightsaber
…	…	

Title	*Length*	*Animator*
Monsters	92	John
…	…	

Title	*Length*
Amadeus	92
…	…

E/R Style:

Title	*Length*
Episode I	133
Monsters	92
…	…

Title	*Weapon*
Episode I	Lightsaber
…	…

Title	*Animator*
Monsters	John
…	…

Using *Nulls*:

Title	*Length*	*Weapon*	*Animator*
Episode I	133	Lightsaber	Null
Monsters	92	Null	John

Relational Algebra

Roadmap

- We created ‘good’ relational schemas applying normalization.
- How do we query the tables to obtain answers?
- We will first define an algebra on sets (well relations are sets!).
- Then we will study an ‘implementation’ of such an algebra in the form of a query language, called SQL (future lectures).

Why define an Algebra?

- Solid theoretical foundation.
- Algebra consists of primitive operators
  - Think of arithmetic algebra!
- Form queries by combining such operators
  - Focus on implementing each operator independently
  - Allows of optimization of expressions written in the algebra
- Enables reasoning about expressiveness of our expressions

“Core” Relational Algebra

A small set of operators that allow us to manipulate relations in limited but useful ways. The operators are:

- *Union*, *intersection*, and *difference*
  … the usual set operators!
  - But the relation schemas must be the same.
- *Selection*: Picking certain rows from a relation
- *Projection*: Picking certain columns
- *Products and joins*: Composing relations in useful ways
- *Renaming* of relations and their attributes

Relational Algebra (cont.)

Characteristics:

- Limited expressive power (subset of possible queries)
- Good optimizer possible
- Rich enough language to express enough useful things

σ stands for SELECT
p stands for PROJECT
} UNARY

× represents CARTESIAN PRODUCT
∪ represents UNION
– represents SET-DIFFERENCE
} BINARY

} FUNDAMENTAL

∩ represents SET-INTERSECTION
$\triangleright\triangleleft_\theta$ represents THETA-JOIN
$\triangleright\triangleleft$ represents NATURAL JOIN
÷ represents DIVISION or QUOTIENT
} Can be defined in terms of the fundamental operations

Union, Intersection, and Difference

R:

name	*address*	*gender*	*birthdate*
Carrie Fisher	123 Maple St.	F	9/9/99
Mark Hamill	456 Oak Rd.	M	8/8/88

S:

name	*address*	*gender*	*birthdate*
Carrie Fisher	123 Maple St.	F	9/9/99
Harrison Ford	789 Palm Dr.	M	7/7/77

R ∪ *S*:

name	*address*	*gender*	*birthdate*
Carrie Fisher	123 Maple St.	F	9/9/99
Mark Hamill	456 Oak Rd.	M	8/8/88
Harrison Ford	789 Palm Dr.	M	7/7/77

Union, Intersection, and Difference (cont.)

R:

name	*address*	*gender*	*birthdate*
Carrie Fisher	123 Maple St.	F	9/9/99
Mark Hamill	456 Oak Rd.	M	8/8/88

S:

name	*address*	*gender*	*birthdate*
Carrie Fisher	123 Maple St.	F	9/9/99
Harrison Ford	789 Palm Dr.	M	7/7/77

R – *S*:

name	*address*	*gender*	*birthdate*
Mark Hamill	456 Oak Rd.	M	8/8/88

R ∩ *S*:

name	*address*	*gender*	*birthdate*
Carrie Fisher	123 Maple St.	F	9/9/99

Note: *R* ∩ *S* = *R* – (*R* – *S*)

Projection

Syntax: $R_1 = \pi_L(R_2)$
where *L* is a list of attributes from the schema of R_2

Example:

Movie:

title	*year*	*length*	*in-Color*	*studio-Name*	*produceC#*
Star Wars	1977	124	true	Fox	12345
Mighty Ducks	1991	104	true	Disney	67890
Wayne's World	1992	95	true	Paramount	99999

Then the result of executing the query $\pi_{title, length, year}(Movie)$ is:

Result:

title	*year*	*length*
Star Wars	1977	124
Mighty Ducks	1991	104
Wayne's World	1992	95

Another example…

Movie:

title	*year*	*length*	*in-Color*	*studio-Name*	*produceC#*
Star Wars	1977	124	true	Fox	12345
Mighty Ducks	1991	104	true	Disney	67890
Wayne's World	1992	95	true	Paramount	99999

Then executing $\pi_{in\text{-}Color}(Movie)$ gives:

Result:

in-Color
true

Selection

Syntax: $R_1 = \sigma_C(R_2)$
where *C* is a condition involving the attributes of R_2.

Example:

Movie:

title	*year*	*length*	*in-Color*	*studio-Name*	*produceC#*
Star Wars	1977	124	true	Fox	12345
Mighty Ducks	1991	104	true	Disney	67890
Wayne's World	1992	95	true	Paramount	99999

Then the result of executing the query $\sigma_{length>=100}(Movie)$ is:

Result:

title	*year*	*length*	*in-Color*	*studio-Name*	*produceC#*
Star Wars	1977	124	true	Fox	12345
Mighty Ducks	1991	104	true	Disney	67890

Another example…

Movie:

title	*year*	*length*	*in-Color*	*studio-Name*	*produceC#*
Star Wars	1977	124	true	Fox	12345
Mighty Ducks	1991	104	true	Disney	67890
Wayne's World	1992	95	true	Paramount	99999

Then executing $\sigma_{length>=100\ \textbf{AND}\ studio\text{-}Name='Fox'}(Movie)$ gives:

Result:

title	*year*	*length*	*in-Color*	*studio-Name*	*produceC#*
Star Wars	1977	124	true	Fox	12345

Cartesian Product

Syntax: $R = R_1 \times R_2$

Semantic: pairs each tuple t_1 of R_1 with each tuple t_2 of R_2 and puts in R a tuple $t_1 t_2$

Example:

R

A	*B*
1	2
3	4

$\times$

S

B	*C*	*D*
2	5	6
•	7	8
9	10	11

=

A	*R.B*	S.B	*C*	*D*
1	2	2	5	6
1	2	4	7	8
•	2	9	10	11
•	4	2	5	6
3	4	4	7	8
3	4	9	10	11

Theta-Join

Syntax: $R = R_1 \triangleright\!\triangleleft_{\theta} R_2$

is equivalent to $R = \sigma_{\theta}(R_1 \times R_2)$

Example:

A	*B*	*C*
1	2	3
6	7	8
9	7	8

$\triangleright\!\triangleleft_{A<D}$

B	*C*	*D*
2	3	4
2	3	5
7	8	10

=

A	*R.B*	R.C	S.B	S.C	*D*
1	2	3	2	3	4
1	2	3	2	3	5
1	2	3	7	8	10
6	7	8	7	8	10
9	7	8	7	8	10

Natural Join

Syntax: $R = R_1 \triangleright\!\triangleleft R_2$

is equivalent to $R = \pi_{R_1 \cup R_2}(\sigma_C(R_1 \times R_2))$

Semantic: Similar to the theta-join of R_1 and R_2 with the condition that all attributes of the same name be equated. Then, one column for each pair of equated attributes is projected out.

Example:

A	*B*
1	2
3	4

$\triangleright\!\triangleleft$

B	*C*	*D*
2	5	6
•	7	8
9	10	11

=

A	*B*	*C*	*D*
1	2	5	6
3	4	7	8

Another Example

A	*B*	*C*
1	2	3
6	7	8
9	7	8

$\triangleright\!\triangleleft$

B	*C*	*D*
2	3	4
2	3	5
7	8	10

=

A	*B*	*C*	*D*
1	2	3	4
1	2	3	5
6	7	8	10
9	7	8	10

Operator Precedence

The normal way to group operators is:

- Unary operators σ, π, and ρ have highest precedence.
- Next highest are the "multiplicative" operators like join, theta join and cross product.
- Lowest are the "additive" operators union, intersection and difference.

But there is no universal agreement, so we always put parentheses *around* the argument of a unary operator, and it is a good idea to group all binary operators with parentheses *enclosing* their arguments.

Example:

Group $R \cup \sigma\ S\ join\ T$ as $R \cup (\sigma(S)\ \text{join}\ T)$.

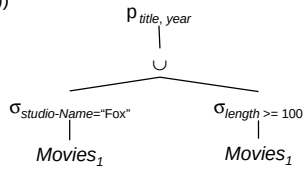
Each Expression Needs a Schema

- If union, intersection and – are applied then the resulting schema is the same with the schema(s) of the arguments.
- Projection: use the attributes listed in the projection.
- Selection: no change in schema.
- Product $R \times S$: use attributes of R and S.
  - But if they share an attribute A, prefix it with the relation name, as $R.A$, $S.A$.
- Theta-join: same as product.
- Natural join: use attributes from each relation; common attributes are merged anyway.
- Renaming

37

Expression trees

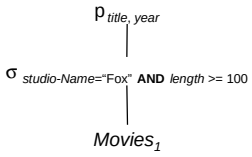
Consider the decomposed relations:
Movies$_1$(*title*, *year*, *length, film-type, studio-Name)*
Movies$_2$(*title, year*, *star-Name*)
Query: "What are the titles and years of movies made by Fox that are at least 100 minutes long"
Answer: $\pi_{title,\ year}(\sigma_{studio\text{-}Name="Fox"}(Movies_1) \cup \sigma_{length >= 100}(Movies_1))$



38

Expression trees (cont.)

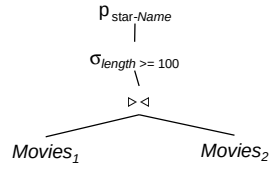
Consider the decomposed relations:
Movies$_1$(*title*, *year*, *length, film-type, studio-Name)*
Movies$_2$(*title, year*, *star-Name*)
Query: "What are the titles and years of movies made by Fox that are at least 100 minutes long"
Answer: $\pi_{title,\ year}(\sigma_{studio\text{-}Name="Fox"\ \textbf{AND}\ length >= 100}(Movies_1))$



39

Expression trees (cont.)

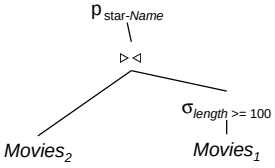
Consider the decomposed relations:
Movies$_1$(*title*, *year*, *length, film-type, studio-Name)*
Movies$_2$(*title, year*, *star-Name*)
Query: "Find the stars of movies that are at least 100 mins long"
Answer: $\pi_{star\text{-}Name}(\sigma_{length >= 100}(Movies_1 \triangleright\triangleleft Movies_2))$



40

Expression trees (cont.)

Consider the decomposed relations:
Movies$_1$(*title*, *year*, *length, film-type, studio-Name)*
Movies$_2$(*title, year*, *star-Name*)
Query: "Find the stars of movies that are at least 100 mins long"
Answer: $\pi_{star\text{-}Name}(Movies_2 \triangleright\triangleleft (\sigma_{length >= 100}(Movies_1)))$



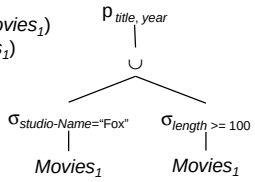
41

Linear Notation for Expressions

- Invent new names for intermediate relations, and assign them values that are algebraic expressions.
- Renaming of attributes implicit in schema of new relation.

Example:

$Tmp_1(t, y, l, f, s) = \sigma_{studio\text{-}Name="Fox"}(Movies_1)$
$Tmp_2(t, y, l, f, s) = \sigma_{length >= 100}(Movies_1)$
$Tmp_3(t, y, l, f, s) = Tmp_1 \cup Tmp_2$
$Answer(title, year) = \pi_{t,y}(Tmp_3)$



42

Bag Semantics

A relation (in SQL, at least) is really a *bag* or a *multiset*.

- It may contain the same tuple more than once, although there is no specified order (unlike a list).
- Example: {1,2,1,3} is a bag and not a set.
- Select, project, and join work for bags as well as sets.
  - Just work on a tuple-by-tuple basis, and don't eliminate duplicates.

43

Bag Union/Intersection/Difference

Union: Sum the times an element appears in the two bags.

- Example: {1,2,1} ∪ {1,2,3,3} = {1,1,1,2,2,3,3}

Intersection: Take the minimum of the number of occurrences in each bag.

- Example: {1,2,1} ∩ {1,2,3,3} = {1,2}.

Difference: Proper-subtract the number of occurrences in the two bags.

- Example: {1,2,1} – {1,2,3,3} = {1}.

44

Laws for Bags and Laws for Sets

- Some familiar laws continue to hold for bags.
  - Examples: union and intersection are still commutative and associative.
- But other laws that hold for sets do not hold for bags.

Example:

$R \cap (S \cup T) \equiv (R \cap S) \cup (R \cap T)$ holds for sets.

- Let R, S, and T each be the bag {1}.
- Left side: $S \cup T = \{1,1\}$; $R \cap (S \cup T) = \{1\}$.
- Right side: $R \cap S = R \cap T = \{1\}$;
  $(R \cap S) \cup (R \cap T) = \{1\} \cup \{1\} = \{1,1\} \neq \{1\}$.

45

Extended ("Non-classical") Relational Algebra

Adds features needed for SQL, bags.

- Duplicate-elimination operator δ.
- Extended projection.
- Sorting operator τ.
- Grouping-and-aggregation operator γ.
- Outerjoin operator join_o

46

Duplicate Elimination

Syntax: $R_2 = \delta(R_1)$

Semantics: Relation R_2 has one copy of each tuple that appears one or more times in relation R_1.

Example:

R_1:

A	B
1	2
3	4
1	2
1	2

$\delta(R_1)$:

A	B
1	2
3	4

47

Sorting

Syntax: $\tau_L(R)$

Semantic: It returns a list of tuples of R, ordered according to attributes on list L.

Note: The result type is outside the normal types (set or bag) for relational algebra.

Example:

R:

A	B
1	3
3	4
5	2

$\tau_B(R) = [(5,2), (1,3), (3,4)]$

48

Extended Projection

Syntax: $R_1 = \pi_L(R_2)$

where L is a list of

- attributes from the schema of R_2
- expressions of the form $x \rightarrow y$, where x, y are attributes of R_2
- expressions of the form $E \rightarrow z$, where E is a *formula* and z a name

Example:

R:

A	B
1	2
3	4

$\pi_{A+B \rightarrow C, A \rightarrow D, A \rightarrow E}(R)$:

C	D	E
3	1	1
7	3	3

Aggregation Operators

- These are not relational operators; rather they summarize a column in some way.
- Five standard operators: SUM, AVG, MIN, MAX and COUNT.

Example:

R:

A	B
1	2
3	4
1	2
1	2

- SUM(B) = 2 + 4 + 2 + 2 = 10
- AVG(A) = (1 + 3 + 1 + 1) / 4 = 1.5
- MIN(A) = 1
- MAX(A) = 3
- COUNT(A) = 4

Grouping Operator

Syntax: $\gamma_L(R)$

where L is a list of elements that are either

- Individual (*grouping*) attributes, or
- Of the form $\theta(A)$, where θ is an aggregation operator and A the attribute to which it is applied

Semantics: $\gamma_L(R)$ is computed by:

- Group R according to all the grouping attributes on list L.
- Within each group, compute $\theta(A)$, on all elements of th group.
- Result is the relation whose columns consist of one tuple for each group. The components of that tuple are the values associated with each element of L for that group and the aggregations over all tuples of that group for the aggregated attributes on list L.

Example

Movie:

title	*year*	*length*	*in-Color*	*studio-Name*	*produceC#*
Star Wars	1977	124	true	Fox	12345
Mighty Ducks	1991	104	true	Disney	67890
Wayne's World	1992	95	true	Paramount	99999
Spider-Man	2002	121	true	Columbia	12345
Episode I	1999	133	true	Fox	45634
Episode II	2002	142	true	Fox	23456

Compute $\gamma_{studio\text{-}Name, AVG(length)}(Movie)$

1. Group by the grouping attribute(s), *studio-Name* in this case:

Star Wars	1977	124	true	Fox	12345
Episode I	1999	133	true	Fox	45634
Episode II	2002	142	true	Fox	23456
Mighty Ducks	1991	104	true	Disney	67890
Wayne's World	1992	95	true	Paramount	99999
Spider-Man	2002	121	true	Columbia	12345

Example (cont.)

2. Compute average of length within groups:

studio-Name	AVG(*length*)
Fox	133
Disney	104
Paramount	95
Columbia	121

Outerjoin

The normal join can "lose" information, because a tuple that doesn't join with any from the other relation (*dangles*) has no vestige in the join result.

- The *null value* $\perp$ can be used to "pad" dangling tuples so that they appear in the join
- Gives us the *outerjoin* operator join_o
- Variations: theta-outerjoin, left- and right-outerjoin (pad only dangling tuples from the left (respectively, right).

Example

R:

A	B
1	2
3	4

S:

B	C
4	5
6	7

$R \circ S$:

A	B	C
3	4	5
1	2	$\perp$
$\perp$	6	7

$R \circ_L S$:

A	B	C
3	4	5
1	2	$\perp$

$R \circ_R S$:

A	B	C
3	4	5
$\perp$	6	7

55