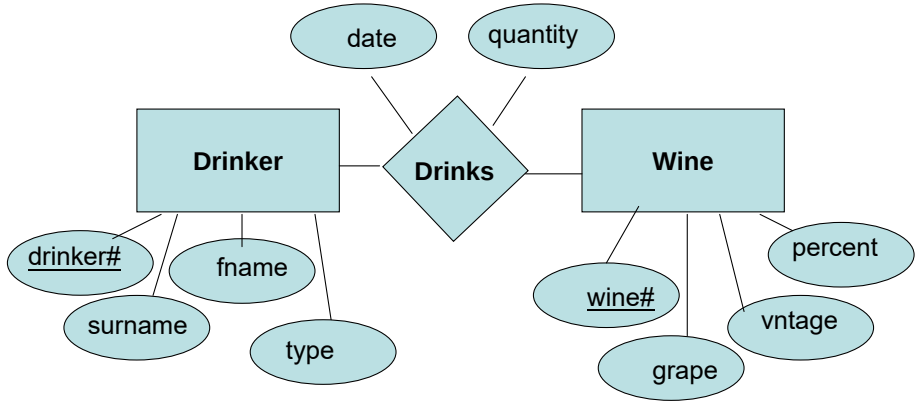
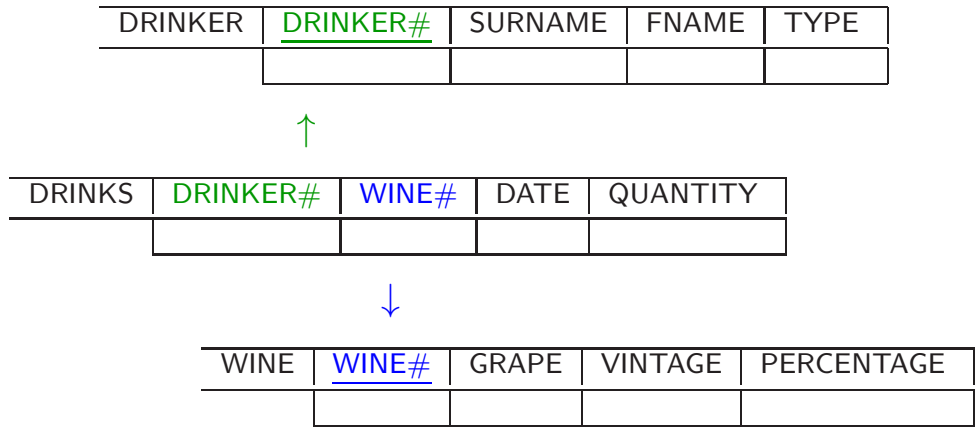
Example: From an ER model to a relational schema



- In this ER model the attribute `dinker#` appears underlined
- This expresses a new kind of semantic constraint: this attribute becomes a unique identifier
- In the sense that any two instances of entity `Drinker` share the same value for attribute `dinker#`, then they have to share the same values for all the other attributes of `Drinker`
- Similarly for `Wine#` in entity `Wine`

- ER model above transformed into a relational schema:



- Arrows indicate referential ICs introduced with the schema

  E.g. any drinker number in relation *Drinks* must appear also in relation *Drinker*

  With the use of shared attributes, we keep relations connected

- We can think relations *Drinker* and *Wine* as the official ones for drinkers and wine, resp.

  Hence, we expect values for them appearing in the *Drinks* relation to also appear in the official ones

- Attribute *Drinker#* appears underlined in relation *Drinker*, indicating a relational constraint (inherited from the ER model)

  In this case, a key constraint: that attribute functionally determines all the others in the relation

  $$Drinker: \ Drinker\# \longrightarrow Surname, \ FName, \ Type$$

- With the expected semantics:

  There cannot be two tuples (rows) in *Drinker* with the same drinker code that have different values for any of the three attributes on the right-hand side (RHS)

- We are not forced to declare the attribute pair $\{Drinker\#, Wine\#\}$ a key for *Drinks*

  A drinker could drink the same wine in different quantities on different dates ...

- Why not a universal relation DRINKING?

Drinking	Drinker#	SURNAME	FNAME	TYPE	WINE#	GRAPE	VINTAGE	PERCENTAGE	DATE	QUANTITY

A broad relation

- In principle possible ...
- Several operational issues when working with populated DB
  - Updates? (insertions, deletions, changes of attribute values)
  - Redundancy of information
  - Identifiers?

    We miss the keys of the ER model

    We could still use (more relaxed) functional dependencies
  - Relational operations with broad relations (tables)
- We will come back to these issues ...

Queries Revisited

- DB instance:

Deposit

branch	acc#	clientn	balance
Carleton	101	Jim	500
Downtown	215	Sandy	700
Barrhaven	304	Alvin	1300

Client

clientn	cladd	neighcl
Jim	101 Queensbury	Barrhaven
Sandy	40 Stone	Nepean
Hernandez	15 Laurier	Downtown
Alvin	17 Clyde	Altavista
John	89 Case	Centrepoint

- *"Give me the balances and addresses of clients whose balance is higher than 600"*
- Answer:

cladd	balance
40 Stone	700
17 Clyde	1300

- The answer is a set of tuples, a new relation (extension)
- We can say that a query is a mapping that sends DB instances to new DB instances (possible with a different schema)

Several issues:

- How to specify a query?

  How to write it?

  In what language?

- How expressive is the chosen query language?

  Can it express "common and useful" queries?

- What is the meaning of a query answer? (semantics)

- How to compute the answer?

- There are several query languages for RDBs

  Some more used in practice than others

- Those of a more theoretic nature are the basis for the languages most used in practice

- The distinction between declarative vs. procedural (imperative) query languages is always relevant
- Declarative languages are used to express what the user wants to obtain from the database
- Procedural languages express a particular way to compute the query answer
- (Similar distinctions apply to programming languages, e.g. Prolog or LISP vs. Python or Java)

Relational Algebra

- Relations are sets (subsets of cartesian products) constructed on top of other sets (domain or subdomains)
- Query answers are new relations
- In order to obtain new relations (e.g. query answers) perform set-theoretic algebra on existing relations
- Operate on sets and relations in order to obtain new sets and relations
- Relational Algebra (RA) as a relational query language allows to express those set-operations on set and relations
- Some of those operations come directly from set theory

  Others are specific, *ad hoc*, for the RA, and are applicable to relations (as opposed to general sets)

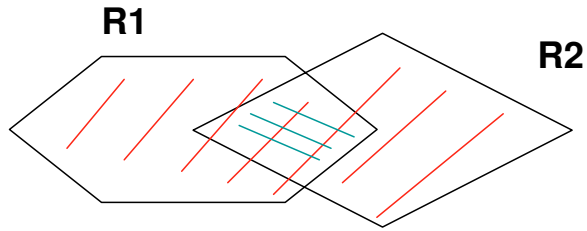
- RA provides a procedural query language for RDBs
- The RA is one of the strengths of the relational model
- RA has a precise, set-theoretic semantics
- A RA query becomes a sequence of algebraic (relational) operations starting from -and applied to- the initial instance
- The sequence becomes an algebraic formula to be executed on the given instance
- RA can be used to provide a semantics to other relational query languages (via translations/compilations)
- RA can be used to compute a query specified in another, e.g. declarative, query language (idem)

  RDBMS offer different query languages, but in the end a query is compiled into a sequence of RA operations on the DB

The Basic RA Operations:

- *Union* and *Intersection*: $R_1 \cup R_2$, $R_1 \cap R_2$

  Applicable to compatible relations as usual sets (relations with the same sub-schema, same arity, same data types)

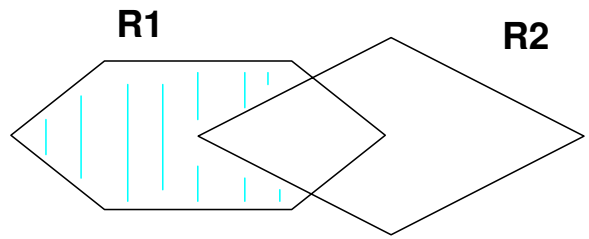


R1 U R2

R1 ⋂ R2

- *Difference*: $R_1 \setminus R_2$

  Applicable to compatible relations as usual sets



R1 \ R2

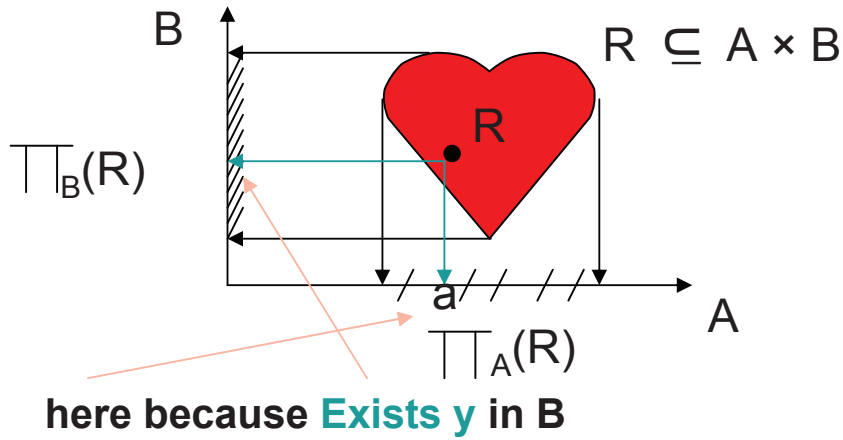
- *Product*: $R_1 \times R_2$

  The cartesian product of two relations (as regular sets), not necessarily compatible

$$R = \{(a, b, c), (c, d, e)\}$$
$$S = \{(1, 2), (2, 3)\}$$
$$R \times S = \{(a, b, c, 1, 2), (a, b, c, 2, 3), (c, d, e, 1, 2), (c, d, e, 2, 3)\}$$

- *Projection*: $\Pi_A R(\cdots, A, \cdots)$

  The projection of relation R on attribute A



- A is one of the attributes of R

  Projection could be on several attributes of R

  A unary operation: takes one relation as input (the previous ones are binary)

- An operation specific for relations (as opposed to general sets)

- It deletes (ignores, filters out) entire "columns" from a relation

  Projects R over one (or several) "coordinates" (attributes)

  It generates a new relation, with a subset of the attributes (columns)

- Example: Schema $R(A, B, C)$

  Extension: $R = \{(e, a, f), (u, c, e)\}$

  $\Pi_{A,C}(R) = \{(e, f), (u, e)\}$

- Its logical counterpart is the existential quantification

  In the figure above:

  $\Pi_A R(A, B, C) = \{a \in A \mid$ there exist $b \in B$ and $c \in C$, such that $(a, b, c) \in R\}$

- *Selection*: $\sigma_{<condition>}(R)$

  Unary operation, special for relations

  Selects tuples from relation R that satisfy the *condition*

  The condition can be expressed in a (limited) logical language

  It generates a new relation, with the same attributes, but possibly fewer tuples (rows)

- Example: $\sigma_{balance>550}(\texttt{Deposit})$

Selects only those tuples of *Deposit* that show a balance greater than 550

Deposit

branch	acc#	clientn	balance
Carleton	101	Jim	500
Downtown	215	Sandy	700
Barrhaven	304	Alvin	1300

branch	acc#	clientn	balance
Downtown	215	Sandy	700
Barrhaven	304	Alvin	1300

- *Join*: $R_1 \bowtie R_2$

A binary operator, essential in RA

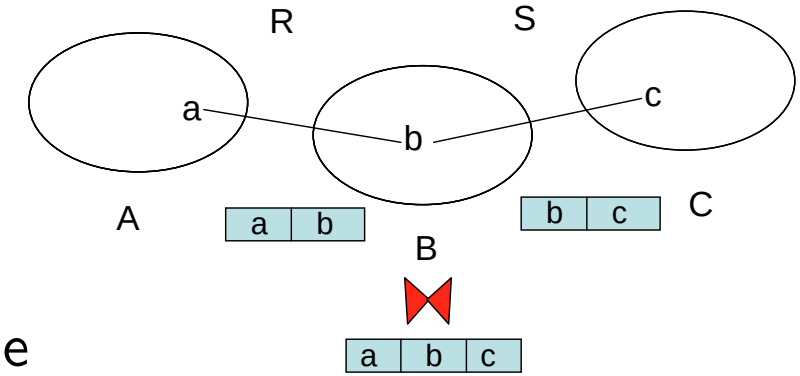
Here, in its simplest form: "natural join"

It allows to compose two relations

Through the values in common taken by a (set of) distinguished attribute(s) in the two relations

Attributes with same data type or domain

- There are extensions of the natural join (later ...)
- Essential for combining tables without explicitly producing the expensive *product* (join is still expensive)



- Example: *"Give me the addresses with balances of the clients who have a balance higher than 600"*

Deposit

branch	acc#	clientn	balance
Carleton	101	Jim	500
Downtown	215	Sandy	700
Barrhaven	304	Alvin	1300

Client

clientn	cladd	neighcl
Jim	101 Queensbury	Barrhaven
Sandy	40 Stone	Nepean
Hernandez	15 Laurier	Downtown
Alvin	17 Clyde	Altavista
John	89 Case	Centrepoint

- RA query: $\Pi_{\text{cladd, balance}}(\sigma_{\text{balance} > 600}(\ \textit{Deposit}\ \bowtie_{\text{clientn}} \textit{Client}\ )$

$=$

40 Stone	700
17 Clyde	1300

- RA query also works:

  $\Pi_{\text{cladd, balance}}(\sigma_{\text{balance} > 600}(\ \textit{Deposit}\ ) \bowtie_{\text{clientn}} \textit{Client}))$

  Which one is "better"?

- This join condition is also possible: (assuming attributes with same data type)

$$Deposit \bowtie_{\text{clientnD} = \text{clientnC}} Client$$

Deposit

branch	acc#	clientnD	balance
Carleton	101	Jim	500
Downtown	215	Sandy	700
Barrhaven	304	Alvin	1300

Client

clientnC	cladd	neighcl
Jim	101 Queensbury	Barrhaven
Sandy	40 Stone	Nepean
Hernandez	15 Laurier	Downtown
Alvin	17 Clyde	Altavista
John	89 Case	Centrepoint

- Join is also based on set-theoretic operation: composition of two relations

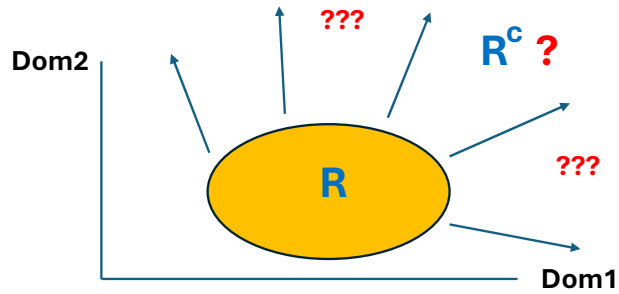
- It could be replaced trough the use of the previous RA operations

$$R(A, B), \quad S(C, D), \text{ with } B, C \text{ of same data type}$$

$R(A, B) \bowtie_{C=D} S(C, D)$ can be defined by $\Pi_{ABD}(\sigma_{B=C}(R \times S))$

- There is no *Complement* operation in RA (as in set theory)
- What is the "meaning" of the complement of a relation?
- *"Tuples that are not in Deposit"*?

  This query is not admissible nor makes sense in RDBs



- It could be infinite since underlying domains may be infinite

  The difference ($\setminus$) is only a *relative complement*