Queries Expressed in RA:

- A query can be expressed as a sequence of operations of RA applied to the original tables and/or intermediate results
- Example: Schemas

DRINKER	DRINKER#	SURNAME	FNAME	TYPE

DRINKS	DRINKER#	WINE#	DATE	QUANTITY

(DATE at day level)

WINE	WINE#	GRAPE	VINTAGE	PERCENTAGE

- Query 1: Percentages of alcohol in Morgon wines, vintage 1979?

$$\begin{aligned} R1 &:= \sigma_{GRAPE=Morgon}(WINE)\\ R2 &:= \sigma_{VINTAGE=1979}(WINE)\\ R3 &:= R1 \cap R2\\ ANS &:= \Pi_{PERCENTAGE}(R3)\end{aligned}$$

- A fixed algebraic query (expression), independent from instance, applicable to any instance; depends only on schema
- $ANS = \Pi_{PERCENTAGE}(\sigma_{GRAPE=Morgon}(WINE) \ \cap \ \sigma_{VINTAGE=1979}(WINE))$

- A procedural (imperative) query: We are telling the system how to compute the desired answers
- Another solution (equivalent to the first one)

  $$ANS = \Pi_{PERCENTAGE}(\sigma_{GRAPE=Morgon\ AND\ VINTAGE=1979}(WINE))$$

- An algebraic formula that can be used to compute the answers

  It can be applied to every particular instance of the DB
- Notice the correspondence between the set-theoretic and logical operations

  Query 2: Last and first names of drinkers of Morgon or Chenas?

  $R1 := \sigma_{GRAPE=Morgon}(WINE)$
  $R2 := \sigma_{GRAPE=Chenas}(WINE)$
  $R3 := R1 \cup R2$
  $R4 := R3 \bowtie_{WINE\#} DRINKS$ ($R3$ is smaller than $WINE$)
  $R5 := R4 \bowtie_{DRINKER\#} DRINKER$ (all attributes of all tables together)
  $ANS := \Pi_{SURNAME,FNAME}(R5)$

- Notice the useful selection before the join

  The other way around would be semantically the same, but less efficient

- Beware: Do not project too early or too much since you may lose information for additional selection/join conditions

  Keep carrying attributes you may need later on

- Query 3: Last and first names of drinkers who have tried in one day more than 10 samples of Chablis, vintage 1976, together with the percentage of alcohol of the wine

$$R1 := \sigma_{QUANTITY>10}(DRINKS)$$
$$R2 := \sigma_{GRAPE=Chablis}(WINE)$$
$$R3 := \sigma_{VINTAGE=1976}(WINE)$$
$$R4 := R2 \cap R3$$
$$R5 := R1 \bowtie_{WINE\#} R4 \qquad \text{(all attributes for } DRINKS \text{ and } WINE \text{ here)}$$
$$R6 := \Pi_{DRINKER\#,PERCENTAGE}(R5) \qquad \text{(keep } DRINKER\# \text{ for next join)}$$
$$R7 := R6 \bowtie_{DRINKER\#} DRINKER$$
$$ANS = \Pi_{SURNAME,FNAME,PERCENTAGE}(R7)$$

- RA is based on set-theoretic operations, i.e. that take and produce sets

  By default, the results do not show duplicates

  That is, no multiple occurrences of the same tuple

- It is possible to extend RA operations to deal with multi-sets or bags

  They may have duplicates

- Exercise: Illustrate the computations for queries 1-3 using a concrete initial instance; and producing all the intermediate relations that lead to the final answer

  Exercise: Schema: $Frequents(Drinker, Bar)$, $Serves(Bar, Beer)$, $Likes(Drinker, Beer)$ Express in RA the following queries:

  1. Which bars serve the beer John likes?
  2. Which drinkers frequent at least one bar that serves some beer they like?
  3. Which drinkers frequent only bars that serve at least one beer they like?
  4. Which drinkers do not frequent any bar that serves some beer they like?

On Query Optimization:

- With RA, to speed up query processing, the system applies products and joins once tables have been reduced using other RA operations (such as intersection, difference, selection and projection reduce relations)
- This Syntactic Query Optimization rearranges a query, as a sequence of RA operations, into a new sequence that leads to less expensive joins

  Still obtaining an equivalent query
- In contrast, for Semantic Query Optimization (see page 10), the original query is rewritten into a new, less expensive query

  The rewriting depends on the syntactic, symbolic interaction of the IC and the original query

  There is a general mechanism that relies on the representation in Relational Calculus (the logical counterpart of RA) of the query and the ICs

- An optimized query resulting from the rearrangement of RA operations is syntactically different but semantically equivalent to the original query

  They have the same semantics (meaning)

- Space is always an issue since DBs can be very large and computations take place in main memory

  RDBMSs have built-in query optimizers that are automatically invoked

- The notion of “semantic equivalence” of queries, in particular of relational expressions, is well-defined and precise

- Two RA queries are equivalent if for every instance of the given schema they produce the same answer

- A strength of RA: The semantics of the language is clear, precise, formal and well-studied

  It is grounded on set theory and predicate logic

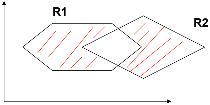
Some Final Remarks:

- There are other RA operations we haven't presented

  In particular, there are other forms of join (later)

- It is possible to define new RA operations on the basis of the already defined operations (and nothing else)

- Example: The "symmetric difference" of two similar relations



$$R1 \ \Delta \ R2 := (R1 \setminus R2) \cup (R2 \setminus R1)$$

Equivalently:

$$R1 \ \Delta \ R2 = (R1 \cup R2) \setminus (R1 \cap R2)$$

- The new operation (Δ) is defined by means of a fixed algebraic formula that uses already defined operations ($\setminus, \cup$)

- A definition applicable to any instance

  That is, the definition is independent from the instance at hand

- Two relations with the same schema

WINE1	W#	GRAPE	VINTAGE	PERCENTAGE
	100	Volnay	1978	12.5
	110	Chablis	1979	12.0
	120	Sancerre	1980	12.5
	130	Tokay	1980	12.5

Δ

WINE2	W#	GRAPE	VINTAGE	PERCENTAGE
	130	Tokay	1980	12.5
	140	Chenas	1981	12.7

=

WINE5	W#	GRAPE	VINTAGE	PERCENTAGE
	100	Volnay	1978	12.5
	110	Chablis	1979	12.0
	120	Sancerre	1980	12.5
	140	Chenas	1981	12.7

- Some of RA operations we introduced can be defined in terms of the others

  They are theoretically redundant (but not necessarily practically redundant)

- For example, the *natural join* can be defined in terms of *product*, *selection*, and *projection* (and possibly the "attribute renaming" operation) Check it!

- There are useful operations on relations that we haven't considered as RA operations
- There is a purely "logical counterpart" to the RA: the Relational Calculus (RC) (see Chapter 1)
- RC is a declarative query language that is based directly on predicate logic
- Example: Query Q1 above can be expressed in RC

  $$Ans(x):\ \exists w\, Wine(w, morgon, 1979, x)$$

  Declaratively expressing what we want, not how to compute it

  We are collecting values for the last attribute, i.e. percentages

  morgon and 1979 are constants from the tables

  The answers to the query are those constants that make the formula true in the DB
- RA or RC query formulation does not require looking into the instance; the schema is good enough

- RA and RC are equivalent in terms of the queries they can express (more on this later)

  They are equally expressive

  Something that can be proved

- Idea of the connection between both:

  Introduce a new logical predicate *Ans* to collect the result

  Next, define it by a logical formula

  1. Selection: $\sigma_{\varphi}(R(A_1, \ldots, A_n))$

     R a relation predicate, and φ a condition on attribute values

     $\forall x_1 \cdots \forall x_n (Ans(x_1, \ldots, x_n) \quad :\longleftrightarrow \quad R(x_1, \ldots, x_n) \wedge \varphi)$

     E.g. $\sigma_{A=a} R(A, B)$ can be defined by

     $\forall x \forall y (Ans(x, y) \quad :\longleftrightarrow \quad R(x, y) \wedge x = a)$

2. Intersection: $R(A, B) \cap S(A, B)$

   $\forall x \forall y (Ans(x, y) \quad :\longleftrightarrow \quad R(x, y) \wedge S(x, y))$

3. Union: $R(A, B) \cup S(A, B)$

   $\forall x \forall y (Ans(x, y) \quad :\longleftrightarrow \quad R(x, y) \vee S(x, y))$

4. Projection: $\Pi_A(R(A, B))$

   $\forall x (Ans(x) \quad :\longleftrightarrow \quad \exists y R(x, y))$

5. Join: $R(A, B) \bowtie_{B=C} S(C, D)$

   $\forall x \forall y \forall z (Ans(x, y, z) \quad :\longleftrightarrow \quad R(x, y) \wedge S(y, z))$

6. Cartesian Product: $R(A, B) \times S(C, D)$

   $\forall x \forall y \forall z \forall w (Ans(x, y, z, w) \quad :\longleftrightarrow \quad R(x, y) \wedge S(z, w))$

7. Difference: $R(A, B) \smallsetminus S(A, B)$

   $\forall x \forall y (Ans(x, y) \quad :\longleftrightarrow \quad R(x, y) \wedge \neg S(x, y))$

- The standard query language for RDBs, SQL, is close to (based on) RC

- A very useful operation we have not considered as a part of RA (or RC) is the Transitive Closure of a binary relation

- Example:

Paternity	Father	Son
	Eric	Luis
	Eric	Juan
	Juan	Carlos
	Juan	Sergio
	Luis	Tomas
	Tomas	Pedro

- We want to define and compute a new relation *Ancestry* that contains all (and only) the tuples that can be obtained by transitive paternity

- *Ancestry* is the transitive closure of Paternity

  The TC is the "smallest" transitive relation that includes Paternity

- "smallest" refers to set-inclusion

  There is not proper subset of Ancestry that is transitive and includes Paternity

Ancestry	Ancestor	Descendant
	Eric	Luis
	Eric	Juan
	Juan	Carlos
	Juan	Sergio
	Luis	Tomas
	Tomas	Pedro
	Eric	Tomas
	Eric	Carlos
	Eric	Sergio
	Luis	Pedro
	Eric	Pedro

- Computation?

  An iterative procedure computes it

Ancestry	Ancestor	Descendant
step 0	Eric	Luis
	Eric	Juan
	Juan	Carlos
	Juan	Sergio
	Luis	Tomas
	Tomas	Pedro
step 1	Eric	Tomas
	Eric	Carlos
	Eric	Sergio
	Luis	Pedro
step 2	Eric	Pedro

- The first step can be computed with $Paternity \bowtie_{Son=Father} Paternity$

  A self-join, obtaining the Grandfather/Grandson relation

  Partial result is joined with *Paternity*, etc., until nothing new

  Each step of the iteration can be computed with a join of RA

- However, the length of the iteration depends on the initial instance

  It is not bounded a priori (for every instance)

- Can we define the TC using a general and fixed formula of RA?

- Theorem: It is not possible to define the TC of a relation by means of a fixed and general formula of RA

- The TC is not part of the RA
- As a consequence, the TC cannot be expressed in RC either

  Actually, one usually proves this first for the RC
- In order to compute the TC in a RDB, an iterative procedure can be programmed in interaction with (or stored in) the DB
- The SQL99 Standard started supporting TC as a query

  Actually, as a recursive view definition

  Recursion is the counterpart of iteration

  We will come back to this ...
- We will have extend RC and RA in different ways

  To be in position to pose (and answer) some common and useful queries

  Some of those extensions will make it into SQL

SQL: Preliminaries

- Recall: Codd's model became widely accepted as the definitive model for RDBMS
- The Structured English Query Language (SEQUEL) developed by IBM Corporation, Inc., to use Codd's model

  SEQUEL became SQL
- 1979: Relational Software Inc. (now Oracle Corporation) introduced Oracle V2

  First commercially available implementation of SQL

  Today it is accepted as the standard language for RDBMS
- The SQL standard created by committees of specialists from companies and universities

  A history of versions of the standard

- Vendors of RDBMSs have their own implementations of SQL
  Loosely following the standard
- SQL is a language with a precise syntax
  Used with/by RDBMs for several tasks:
  - For manipulating data and metadata in a RDBMS
  - Creation and modification of schemas
  - Population of a DB
  - Insertion, modification or deletion of tuples
  - Declaration of ICs (some)
  - Formulation of queries to the DB
    Mostly declarative
  - Definition of views
  - Creation of triggers and stored procedures

- Main (original) purpose: Querying data

  Most of the query part of SQL can be translated into relational calculus (or RA)

  From where it gets a precise semantics (mostly)
- Operations on data are internally compiled into RA operations
- Allows to work with data at the logical level

  Specifying conditions that data must satisfy
- SQL query commands are taken by the query optimization modules of the DBMS

  They determine the best way to access and process the specified data

SQL: Initial Declarations

Defining and Populating a Database Schema:

- "`CREATE TABLE` name (list of elements)"

  Principal elements are attributes and their types

  Also declarations of key and constraints

  "`DROP TABlE` deletes the created relation element

- Example:

```
CREATE TABLE Sells (
        bar CHAR(20),
        beer VARCHAR(20),
        price REAL );
```

```
DROP TABLE Sells;
```

- Data Types:
  1. `INT` or `INTEGER`
  2. `REAL` or `FLOAT`
  3. `CHAR(n)` = fixed length character string
  4. `VARCHAR(n)` = variable-length strings up to `n` characters
  5. DATE. SQL form is `DATE 'yyyy-mm-dd'`
  6. TIME. Form is `TIME 'hh:mm:ss[.ss...]'`

Etc.

Declaring Keys:

- Use `PRIMARY KEY` or `UNIQUE`

  But only one primary key, many UNIQUEs allowed

- SQL instructs implementations to create an index in response to `PRIMARY KEY` only

  (data structure to speed up access given a key value)

  Oracle and DB2 create them for both (i.e. also for "uniques")

- SQL does not allow nulls in a primary key

  But allows them in "unique" columns (with some restrictions)

- Two places to declare keys:
  - After an attribute's type, if the attribute is the key
  - As a separate element if key has more than one attribute

- Example: Relation schemas:

Bars(<u>name</u>, addr, license)

```
CREATE TABLE Bars (
       name CHAR(20) PRIMARY KEY,
       addr VARCHAR(20),
       licence VARCHAR(20)
       );
```

Sells(<u>bar</u>, <u>beer</u>, price)

```
CREATE TABLE Sells (
       bar CHAR(20),
       beer VARCHAR(20),
       price REAL,
       PRIMARY KEY(bar,beer)
       );
```

- These are different:

```
CREATE TABLE Sells (
       bar CHAR(20),
       beer VARCHAR(20),
       price REAL,
       UNIQUE(bar,beer)
       );
```

```
CREATE TABLE Sells (
       bar CHAR(20) UNIQUE,
       beer VARCHAR(20) UNIQUE,
       price REAL,
       );
```

Referential ICs and Foreign Keys:

- Example: Attribute `name` is primary key in `Beers`

  Beers sold in table `Sells` must appear in the table of beers

```
CREATE TABLE Beers (
       name CHAR(20) PRIMARY KEY,
       manf CHAR(20));

CREATE TABLE Sells (
       bar CHAR(20),
       beer CHAR(20) REFERENCES Beers(name),
       price REAL);
```

  `beer` (in `Sells`) indirectly declared as foreign key of `Sells`

  Referencing/pointing to `Beers.name`

- Explicit alternative: Add a new declaration element

```
CREATE     TABLE Sells (
        bar CHAR(20),
        beer CHAR(20),
        price REAL,
        FOREIGN KEY beer REFERENCES Beers(name));
```

```
CREATE     TABLE Sells (
        bar CHAR(20),
        beer CHAR(20),
        price REAL,
        FOREIGN KEY beer REFERENCES Beers(name));
```

- This alternative is essential if the primary key (in the other table) contains more than one attribute
- `Sells.beer` is allowed to take null values without forcing `Beers.name` to have them

  As expected for a primary key

  This is allowed still satisfying the created RIC
- When is a FKC (or RIC) violated?

  What can be done?

  What does the RDBMS does (or can do)?

- Two kinds of violation:

1. Insertion or update of a tuple in `Sells` that refers to a non-existing `beer` in `Beer.name`

   Transaction is always rejected

2. Deletion or update of a tuple in `Beers` that is being referenced by a tuple in `Sells.beer`

   - Default: transaction is rejected

   - Cascade effect: Propagate the changes to tuples in `Sells` that make reference to the updated tuples in `Beers`

   - Set `NULL` in all referencing tuples in `Sells`

- Example: (a) Deletion of `Bud` from official table `Beers`: Delete from `Sells` all tuples containing Bud

  (b) Update of `Bud` to `Budweiser` in `Beers`: Change in all tuples in `Sells`, `Bud` by `Budweiser`

- Example: (a) Deletion of `Bud` in `Beers`: All tuples in `Sells` that had `Bud` now have `NULL` instead

  Those nulls do not have to appear in `Beers.name` to satisfy the RIC

  (b) Update of `Bud` to `Budweiser`: Same change

- With last two cases in Item 2. above, some changes are triggering other changes

  With the purpose of maintaining the consistency of the DB

- Can we specify any of these IC maintenance policies?

- With FKC declaration: `ON [DELETE, UPDATE] [CASCADE, SET NULL]`

- Example:

```
CREATE TABLE Sells (
        bar CHAR(20),
        beer CHAR(20),
       price REAL,
       FOREIGN KEY beer REFERENCES Beers(name)
       ON DELETE SET NULL
       ON UPDATE CASCADE );
```

(this is application dependent)