4.6 The Domain Relational Calculus

- non-procedural, declarative query language

Queries in the domain relational calculus

- variables are bound to domains, i.e., to value sets of attributes
- form: \{[v_1, ..., v_n] \mid P(v_1, ..., v_n)\}
- the $v_i$ ($1 \leq i \leq n$) are **domain variables** representing an attribute value, $P$ is a predicate (a formula) containing the free variables $v_1, ..., v_n$.
- single variables are not bound to a relation, but a list of domain variables is bound to a relation
- a formula is composed of **atoms**, an atom has the following form:
  - $[v_1, ..., v_n] \in R$ $R$ $n$-ary relation, assignment of the $n$ domain variables $v_1, ..., v_n$ to the attributes of $R$ according to the order of the attributes in the schema
  - $x \theta y$ $x$ and $y$ are domain variables, $\theta \in \{=, \neq, <, \leq, >, \geq\}$, $\theta$ must be applicable to the domain
  - $x \theta c$ $x$ domain variable, $c$ is constant with $c \in dom(x)$, $\theta$ comparison operator, must be applicable to the domain
structure of formulae (bottom-up approach)
- All atoms are formulae.
- If \( P \) is a formula, then so are \( \neg P \) and \( (P) \).
- If \( P \) and \( Q \) are formulae, then so are \( P \land Q \), \( P \lor Q \) and \( P \Rightarrow Q \).
- If \( P(v) \) is a formula containing a free variable \( v \), then \( \exists v(P(v)) \) and \( \forall v(P(v)) \) are also formulae.

shorter notation: e.g. \( \exists v_1, v_2, v_3(P(v_1, v_2, v_3)) \) instead of \( \exists v_1(\exists v_2(\exists v_3(P(v_1, v_2, v_3)))) \)

query example: Determine the registration ids and names of students who have passed at least one exam from professor Curie.
- \( \{[m, n] \mid \exists s([m, n, s] \in \text{students} \land \exists v, p, g([m, v, p, g] \in \text{tests} \land \exists a, r, b([p, a, r, b] \in \text{professors} \land a = \text{"Curie"}))) \}
- Join conditions are implicitly specified by using the same domain variables. In the example the variable \( m \) is used to perform the join between \text{students} and \text{test}. Variable \( m \) represents in both tuples \([m, n, s] \in \text{students} \) and \([m, v, p, g] \in \text{tests} \) the same registration id.

relational algebra, tuple relational calculus, and domain relational calculus have the same expressive power.
structure of formulae (bottom-up approach)
- All atoms are formulae.
- If \( P \) is a formula, then so are \( \neg P \) and \( (P) \).
- If \( P \) and \( Q \) are formulae, then so are \( P \land Q \), \( P \lor Q \) and \( P \Rightarrow Q \).
- If \( P(v) \) is a formula containing a free variable \( v \), then \( \exists v(P(v)) \) and \( \forall v(P(v)) \) are also formulae.

shorter notation: e.g. \( \exists v_1, v_2, v_3(P(v_1, v_2, v_3)) \) instead of \( \exists v_1(\exists v_2(\exists v_3(P(v_1, v_2, v_3)))) \)

query example: Determine the registration ids and names of students who have passed at least one exam from professor Curie.
- \( \{[m, n] | \exists s([m, n, s] \in \text{students} \land \exists v, p, g([m, v, p, g] \in \text{tests} \land \exists a, r, b([p, a, r, b] \in \text{professors} \land a = \text{"Curie"}))) \}
- Join conditions are implicitly specified by using the same domain variables. In the example the variable \( m \) is used to perform the join between \text{students} and \text{test}. Variable \( m \) represents in both tuples \([m, n, s] \in \text{students} \) and \([m, v, p, g] \in \text{tests} \) the same registration id.

relational algebra, tuple relational calculus, and domain relational calculus have the same expressive power.
4.7 Extension of the relational algebra on the basis of multi-relations

Problems of the relational algebra and the relational calculus

- Since the relational algebra is a universal algebra, some important query types are not supported by the model.

- Sorting of data (especially as the result of a query) desirable

- Storing of duplicates, which, e.g., have been computed by the projection operation, is frequently desired

- Insufficient functionality of the relational algebra: aggregation (e.g., sum, average, maximum) of the data of a relation desirable but not enabled by the model

- In SQL, which is the standard database query language for relational systems, these design requirements have been taken into account.
5. SQL - the Relational Database Language Standard

5.1 Introduction

Most relevant query languages

- development of special languages for relational DBMS, based on tuple relational calculus and relational algebra
- SQL (Structured Query Language) is the most popular database language
- also of practical importance: QBE (Query by Example)
- the language Quel (Query Language) was developed for the DBMS Ingres, did not prevail over SQL

SQL

- developed 1974 at IBM as language of the relational DBMS System R
- SQL can be regarded as a hybrid between an extended relational algebra and the relational calculus. SQL is a language standard now.
- versions: SQL1 (1985), SQL2 (1992, also denoted as SQL92), SQL3 (1999, also denoted as SQL:1999), in this chapter: excerpt from SQL2
Components of SQL

- **data definition language (DDL)**
  - creation and change of the data structures for the three levels of a database (external levels, conceptual level, physical level): definition of relation schemas, deletion of relations, creation of indexes, modification of relation schemas, creation of views
  - specification of integrity constraints
  - fixing of access rights (authorization)

- **data manipulation language (DML)**
  - insertion, change and deletion of data objects
  - interactive formulation of queries

- **embedded DML**
  - embedding of SQL-commands into an all-purpose programming language (host language) like e.g. Fortran, C, C++ or Java

- **transaction control**
  - commands for specifying the begin, abort or end of transactions, in some implementations explicit commands for locking data for concurrency control
5.2 Data definition language (DDL)

Data types

- primarily numbers, strings and date declarations as fundamental data types for attribute domains

- in detail:
  - **char**\(n\) character string of fixed length \(n\), with user specified length \(n\), synonym: **character**\(n\)
  - **varchar**\(n\) character string of variable length, with user specified maximum length \(n\), synonym: **char varying**\(n\), **character varying**\(n\)
  - **int** integer, value of a computer-dependent, finite subset of the whole numbers, synonym: **integer**
  - **smallint** small integer, a computer-dependent subset of the **int**-domain
  - **numeric**\(z, n\) fixed-point (decimal) number with user specified precision, \(z = \) total number of digits, \(n = \) number of the \(z\) digits to the right of the decimal point, synonym: **decimal**\(z, n\)
  - **real** floating-point number with computer-dependent precision
- **double precision**  double-precision floating-point number with computer-dependent precision
- **float(n)**  floating-point number with user specified precision of at least $n$ digits
- **bit(n)**  bit string of fixed length $n$
- **bit varying(n)**  bit string of variable length with user specified maximum length $n$
- **blob**  **binary large object**, byte sequence of variable length up to 4 GB, for the representation of extremely large objects (e.g. multimedia objects, video sequences, geo-objects)
- **date**  calendar date with year (4 digits), month (2 digits), day (2 digits), format: YYYY-MM-DD
- **time**  time of day, in hours, minutes, and seconds, format: HH:MM:SS
- **time with time zone**  time difference to GMT (6 digits)
- **timestamp**  value containing date and time of date
- **interval**  relative value which can increment or decrement an absolute value of type **date, time** or **timestamp**, year/month- or day/hour-intervals
(restricted) declaration of a domain

- advantage: simple change of a data type for a domain which is used from several attributes in a schema
- form: `create domain < my type> as < type specification >`

  example: `create domain string as varchar(256)`

Specification of integrity constraints and default values

- Since SQL allows null values (**null**), an integrity constraint **not null** can be defined, if for a specific attribute a null value is **not** allowed.
- It is recommended to specify this condition for each primary key.
- definition of a default value for an attribute by attaching the clause **default** `< value >` to the attribute definition
- The default value is inserted into each new tuple, if an explicit value for this attribute is not specified. If a default clause is not defined, the default value is **null**.
- The clause **primary key** specifies one or more attributes that form the primary key of the relation.
- definition of a foreign key by the **foreign key** clause (**referential integrity**)