Extensible data types

- Informix Universal Server = basic DBMS + Data Blade Modules
- Number of pre-defined data types
  - types for two-dimensional geometric objects (points, lines, circles, ellipses), images, time series, text, Web pages
  - when the Informix Universal Server was announced, 29 Data Blades existed
- Application may create own types, thus making the data type notion fully extendible.
- four constructs to declare additional data types
  - opaque type
  - distinct type
  - row type
  - collection type
- Opaque type
  - Internal representation is hidden and used for encapsulating a type.
  - User has to provide casting functions to convert an opaque object between its hidden representation in the server data base and the visible representations on the client.
− User functions `send/receive` to convert the internal server representation to/from the client representation
− Specification of an opaque type includes name, fixed internal length, maximal variable length, alignment (which is the byte boundary), hashing possible or not
  
  ```sql
  create opaque type fixed_opaque_udt (internallength = 8, alignment = 4, cannothash);
  create opaque type var_opaque_udt (internallength = variable, maxlen = 1024, alignment = 8);
  ```

- Distinct type
  
  − An existing type is extended through inheritance. The newly defined type inherits the functions/methods of its base type, if they are not overwritten.
    
    ```sql
    create distinct type hiring_date as date;
    ```

- Row type
  
  − represents a composite attribute, analogous to a `struct` type in C
  − support of inheritance by using the keyword `under`
    
    ```sql
    create row type employee_type(ename varchar(25), address varchar(200), salary int);
    create row type engineer_type(licence varchar(20)) under employee_type;
    ```
create row type engr_mgr_type(manager_start_date varchar(10), dept_managed varchar(20)) under engineer_type;

- application, e.g. `create table employee of type` employee_type;

- Collection type
  - Collections include lists (`list`), sets (`set`) und multisets (`multiset`) of built-in types and user-defined types.
  - Example
    ```sql
    create table employee(name varchar(50) not null, commission multiset(money));
    ```

Support of user-defined routines

- user-defined functions to manipulate user-defined types
- Implementation as stored procedures or in C or in Java
- Example:
  ```sql
  create function equal(arg1 fixed_opaque_udt, arg2 fixed_opaque_udt)
  returning boolean;
  external name "/usr/lib/informix/libopaque.so (fixed_opaque_udt_equal)"
  language C;
  end function;
  ```
Support for inheritance

- two kinds of inheritance
  - data (attribute) inheritance
  - function (operation) inheritance

- Data inheritance
  - creation of subtypes with the `under` keyword
  - enable the specification of type hierarchies
  - subtype inherits all attributes of the superior types up to the root

- Example

  ```sql
  create row type employee_type (ename varchar(25); ssn char(9); salary int);
  create row type engineer_type (degree varchar(10); licence varchar(20)) under employee_type;
  create row type engr_mgr_type (start_date varchar(10); dept_managed varchar(20)) under engineer_type;
  ```

  Type `engineer_type` is subtype of type `employee_type`, inherits all attributes of `employee_type`, and has two additional attributes.
Function inheritance
  - Functions can also be inherited.
  - Example:
    ```sql
    create function overpaid(employee_type)
    returns boolean as
    return $1.salary > (select salary from employee where ename = "Bill Brown")
    ```
  - All tables under the `employee` table automatically inherit this function.
  - Redefinition for `engr_mgr_type`:
    ```sql
    create function overpaid(engr_mgr_type)
    returns boolean as
    return $1.salary > (select salary from employee where ename = "Jack Jones")
    ```
  - The query
    ```sql
    select g.ename from engineer as g where overpaid(g);
    ```
  - This is called **operation (function) overloading** (special kind of polymorphism).
Support for index extensions

- Example:
  
  The query
  
  ```sql
  create index empl_city on employee (city(address));
  ```
  
  creates an index on the table `employee` using the value of the `city` function.

- In order to support user-defined index structures, operator classes are available that support user-defined data types in a generic B-tree.

- R-trees are available for geometric data types.

Support for Data Blades API

- purpose: definition of new data types and functions for new kinds of applications

- Two-dimensional spatial data types for geometric applications, e.g.
  
  - A `point` is defined by (X, Y) coordinates.
  - A `line` is defined by its two end points.
  - A `polygon` is defined by an ordered list of n points that form its vertices.
  - A `path` is defined by a sequence (ordered list) of points.
  - A `circle` is defined by its center point and its radius.
Functions on these data types comprise e.g.

- distance: $point \times point \rightarrow float$
- distance: $point \times line \rightarrow float$
- distance: $point \times polygon \rightarrow float$
- distance: $line \times line \rightarrow float$
- ...
- overlap: $polygon \times polygon \rightarrow bool$
- overlap: $line \times polygon \rightarrow bool$
- ...
- contains: $point \times polygon \rightarrow bool$
- contains: $point \times circle \rightarrow bool$
- ...

Image data types

- Data type $image$ offered with a large variety of standard storage formats, support of the formats tiff, gif, jpeg, photoCD, group 4, fax
- Operations e.g.
  
  rotate: $image \times angle \rightarrow image$
  crop: $image \times polygon \rightarrow image$
  flip: $image \rightarrow image$
  plus: $image \times image \rightarrow image$
minus: $image \times image \rightarrow image$
intersection: $image \times image \rightarrow image$
union: $image \times image \rightarrow image$

- further data types e.g. for time series, text
**Question 1:** Let $R(A,B,C,D)$ be a relation schema in 1NF. Let $AB$ be the primary key (PK) of $R$. Determine a set of FDs so that $R$ is not in 2NF.

**Solution:** Consider the set of FD: $AB \rightarrow CD$ and $B \rightarrow C$. The non-prime attribute $C$ is not fully functionally dependent on the primary key $AB$. Hence, it is not in 2NF.
Question 2: Let \( R(A, B, C, D) \) be a relation schema in 2NF. Let \( AB \) be the primary key (PK) of \( R \). Determine a set of FDs so that \( R \) is not in 3NF.

Solution: Consider the set of FD: \( AB \rightarrow CD \) and \( C \rightarrow D \). We observe:

1. \( AB \) is a superkey since \( AB \rightarrow R \).
2. \( AB \) is a primary key since \( A^+ \neq R \) and \( B^+ \neq R \).
3. \( R \) is in 2NF since the non-prime attributes \( C \) and \( D \) are fully functionally dependent on the PK \( AB \).
4. \( R \) violates the 3NF because
   a) \( \{D\} \subseteq \{C\} \) is false, that is, \( C \rightarrow D \) is not a trivial FD
   b) The statement “\( C \) is a superkey” is false.
   c) The statement “\( D \) is part of some candidate key” is false.