Advanced Management of Data

Object-Relational Database Systems (2)

Extensions of ORDBMSs

Extensions of ORDBMSs to overcome some limitations of RDBMS:

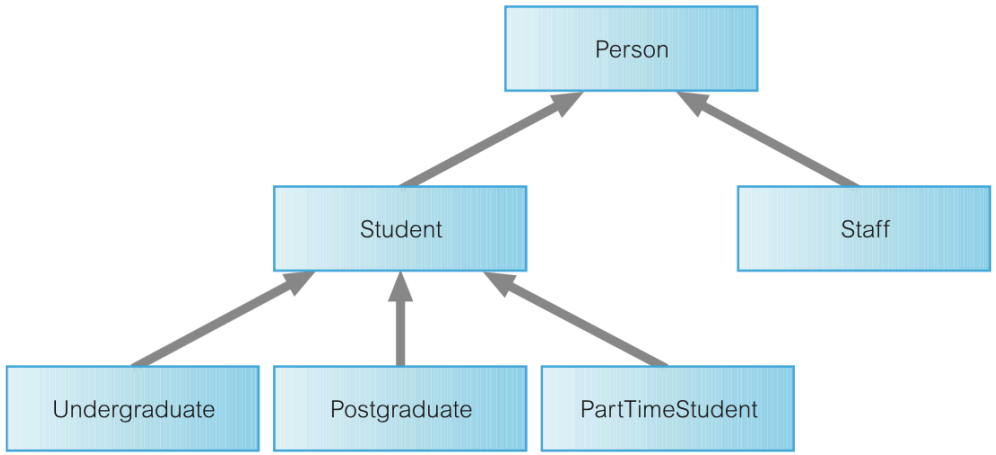
- row types and reference types
- user-defined types
- supertype / subtype relationships
- user-defined procedures, functions, and methods
- table inheritance
- collection types (arrays, sets, lists, multisets)
- large objects

Some of these extensions of SQL:2011 may be implemented differently in some ORDBMSs or not implemented at all.

Type Inheritance

Type Hierarchy

- a subtype inherits all the attributes and methods of its supertype
- a subtype can define additional attributes and methods like any other UDT and it can override inherited methods
- Notice that tables which use types in a supertype/subtype relationship do not automatically provide instance inheritance

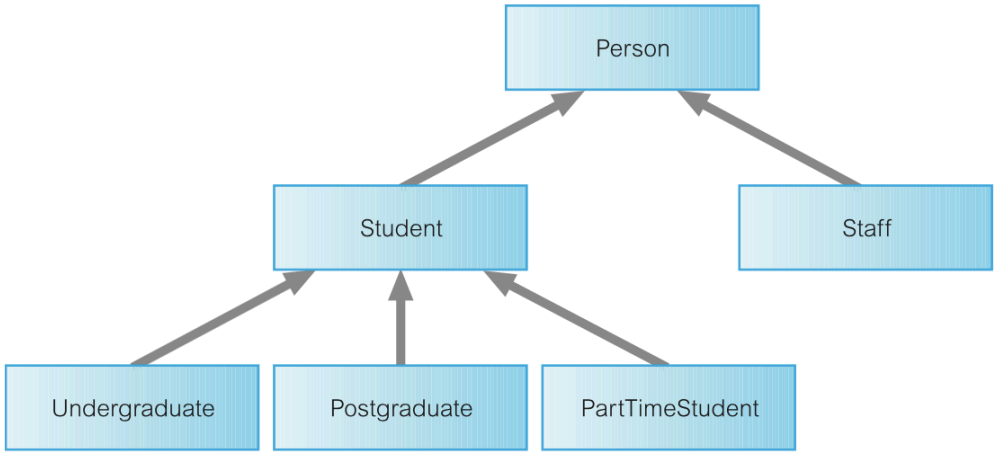


[Connolly & Begg]

Type Inheritance

Example

- Lets create a table P of type *Person* and a table S of type *Student*:
  - If we insert a new row into table S, this row is only inserted into S
  - if we want this row to be inserted into P automatically, we have to use Table Inheritance



[Connolly & Begg]

Table Inheritance

Table Inheritance is realized with the `UNDER`-clause of a `CREATE TABLE` statement.

Semantics

When a row is

- inserted into a subtable, then the values of any inherited columns are inserted into the corresponding supertables, cascading upwards in the table hierarchy
- updated in a subtable, the values of inherited columns in the supertables are updated similarly
- updated in a supertable, then the values of all inherited columns in all corresponding rows of its direct and indirect subtables are also updated accordingly
- deleted in a subtable/supertable, the corresponding rows in the table hierarchy are deleted

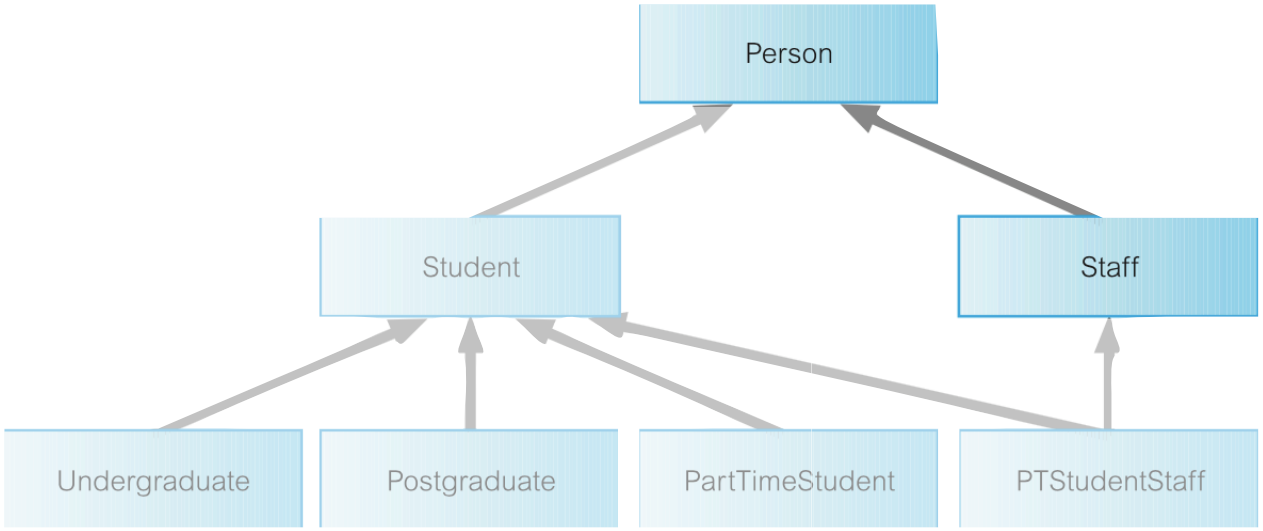
Example

Creation of a subtable

```
CREATE TABLE Staff OF StaffType UNDER Person;
```

Restrictions on the population of a table hierarchy:

- Each row of the supertable *Person* can correspond to at most one row in *Staff*.
- Each row in *Staff* must have exactly one corresponding row in *Person*.



[Connolly & Begg]

Creating Tables

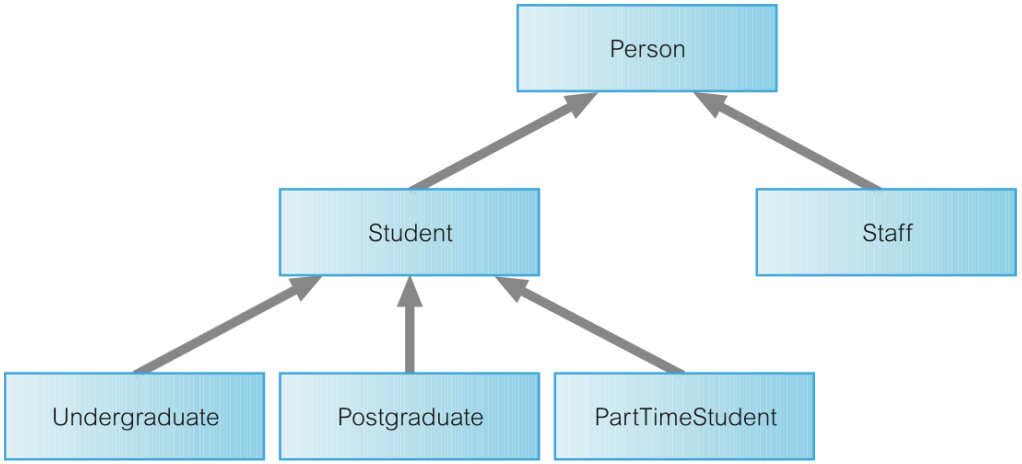
Storing all instances of a UDT

SQL:2011 does not provide a mechanism to store all instances of a given UDT unless the user explicitly creates a single table in which all instances are stored.

Thus, it may not be possible to apply an SQL query to all instances of a given UDT.

Example

```
CREATE TABLE Client (
  info     PersonType,
  prefType CHAR,
  maxRent  DECIMAL(6,2),
  branchNo VARCHAR(4) NOT NULL);
```



[Connolly & Begg]

Now the instances of *PersonType* are distributed over two tables: *Staff* and *Client*.

Querying Data

Retrieving specific columns / rows

- Example - Find the names of all Managers.

```
SELECT s.lName
FROM   Staff s
WHERE  s.position = 'Manager';
```

Invoking user-defined functions

- Example - Find the names of all Managers.

```
SELECT s.lName
FROM   Staff s
WHERE  s.isManager;
```

```
CREATE TYPE StaffType UNDER PersonType AS (
    staffNo  VARCHAR(5),
    position VARCHAR(10),
    salary   DECIMAL(7,2),
    branchNo CHAR(4))
                        …
INSTANCE METHOD isManager () RETURNS BOOLEAN;

CREATE INSTANCE METHOD isManager() RETURNS
BOOLEAN
FOR StaffType
BEGIN
    IF SELF.position = 'Manager' THEN
        RETURN TRUE;
    ELSE
        RETURN FALSE;
    END IF
END;

CREATE TABLE Staff OF StaffType UNDER Person;
```

Querying Data

Use of `ONLY` to restrict selection

- Example - Find the names of all people in the database over 65 years of age.

```
SELECT p.lName, p.fName
FROM   Person p
WHERE  p.age > 65;
```

- Example - Find the names of all people in the database over 65 years (excluding subtables)

```
SELECT p.lName, p.fName
FROM   ONLY (Person) p
WHERE  p.age > 65;
```

```
CREATE TYPE PersonType AS (
    dateOfBirth DATE,
    fName       VARCHAR(15),
    lName       VARCHAR(15),
    sex         CHAR)
                        …
INSTANCE METHOD age () RETURNS INTEGER,
INSTANCE METHOD age (dob DATE) RETURNS
PersonType;

CREATE INSTANCE METHOD age () RETURNS INTEGER
                        …
CREATE INSTANCE METHOD age (dob DATE) RETURNS
PersonType
                        …
CREATE TABLE Person OF PersonType (…)
CREATE TABLE Staff OF StaffType UNDER Person;
```

Querying Data

Use of the dereference operator

References can be used in path expressions that permit traversal of object references to navigate from one row to another by using the dereference operator (–>).

Example - Find the name of the member of staff who manages property 'PG4'.

```
SELECT p.staffID–>fName AS fName, p.staffID–>lName AS lName
FROM   PropertyForRent p
WHERE  p.propertyNo = 'PG4'';
```

➔ **No join or subquery needed!**

```
CREATE TABLE PropertyForRent(
    propertyNo VARCHAR(5) NOT NULL,
                    …
    staffID    REF(StaffType) SCOPE Staff
               REFERENCES ARE CHECKED
               ON DELETE CASCADE,
    PRIMARY KEY (propertyNo)  );
```

```
CREATE TYPE PersonType AS (
    fName   VARCHAR(15),
    lName   VARCHAR(15),
              …
CREATE TYPE StaffType
  UNDER PersonType AS (
    staffNo VARCHAR(5),
              …
```

Querying Data

Use of the dereference operator

Example - Retrieve the member of staff (entire row) for property 'PG4':

```
SELECT DEREF(p.staffID) AS Staff
FROM PropertyForRent p
WHERE p.propertyNo = 'PG4';
```

```
CREATE TABLE PropertyForRent(
    propertyNo VARCHAR(5)     NOT NULL,
    …
    staffID    REF(StaffType) SCOPE Staff
               REFERENCES ARE CHECKED
               ON DELETE CASCADE,
    PRIMARY KEY (propertyNo)  );
```

```
CREATE TYPE PersonType AS (
    fName   VARCHAR(15),
    lName   VARCHAR(15),
              …
CREATE TYPE StaffType
  UNDER PersonType AS (
    staffNo VARCHAR(5),
              …
```

Collection Types

Collections

- are type constructors that are used to define collections of other types
- are used to store multiple values in a single column of a table and can result in nested tables where a column in one table actually contains another table.
- the element type of a collection type constructor may be
  - a predefined type
  - a UDT
  - a row type
  - another collection
- the element type of a collection type constructor cannot be a reference type or a UDT containing a reference type

Collection Types

Collections

Each collection must be homogeneous: All elements must be from the same type hierarchy.

Following collection types exist:

- `ARRAY` one-dimensional array with a maximum number of elements (ordered collection with duplicates allowed)
- `LIST` ordered collection (duplicates allowed)
- `MULTISET` unordered collection (duplicates allowed)
- `SET` unordered collection without duplicates

Collection Types

Array

An array is an ordered collection of values, whose elements are referenced by their ordinal position in the array.

An array is declared by a data type and optionally a maximum cardinality, e.g.

```
VARCHAR(25) ARRAY[5]
```

The elements of an array can be accessed by an index ranging from 1 to the maximum cardinality.

The function `CARDINALITY` returns the number of current elements in the array.

Identity of arrays

Two arrays of comparable types are considered identical if they have the same cardinality and every ordinal pair of elements is identical.

Collection Types

Array

An array type is specified by an array type constructor, which can be defined by

- enumerating the elements as a comma-separated list enclosed in square brackets
- using a query expression with degree 1

Examples:

```
ARRAY ['Mary Blue', 'Ian Beech', 'Ann Ford', 'John Howe', 'Alan Brand']
```

```
ARRAY (SELECT rooms FROM PropertyForRent)
```

In these cases, the data type of the array is determined by the data types of the various array elements.

Example

Use of an ARRAY collection

To model the requirement that a branch has up to three telephone numbers, we could implement the column as an `ARRAY` collection type:

```
telNo VARCHAR(13) ARRAY[3]
```

To retrieve the first and last telephone numbers at branch 'B003' can be done using the following query:

```
SELECT telNo[1], telNo[CARDINALITY(telNo)]
FROM   Branch
WHERE  branchNo = 'B003';
```

Collection Types

Multiset

- is an unordered collection of elements, all of the same type, with duplicates permitted
- is an unbounded collection with no declared maximum cardinality
- operators are provided to convert
  - a multiset to a table (`UNNEST`)
  - a table to a multiset (`MULTISET`).

Set

- a set is a special kind of multiset: one that has no duplicate elements
- a predicate is provided to check whether a multiset is a set

Collection Types

Identity of Multisets

Two multisets of comparable element types A and B are considered identical if they have the same cardinality and for each element *x* in A, the number of elements of A that are identical to *x* (including *x* itself) equals the number of elements of B that are equal to *x*.

Multiset type constructor

Multisets can be defined by

- enumerating elements as a comma-separated list enclosed in square brackets
- using a query expression with degree 1
- using a table value constructor

Collection Types

Operations on Multisets

- `SET` removes duplicates from a multiset to produce a set
- `CARDINALITY` returns the number of current elements
- `ELEMENT` returns
  - the element of a multiset if the multiset only has one element
  - `NULL` if the multiset has no elements
  - An exception is raised if the multiset has more than one element

Example

Use of a collection MULTISET

- Query: Extend the Staff table to contain the details of a number of relatives:

  We include the definition of a *relatives* column in *Staff*:

```
relatives NameType MULTISET
```

- Query: Find the first and last names of John White's relatives:

```
SELECT n.fName, n.lName
FROM   Staff s,
       UNNEST (s.relatives) AS n(fName, lName)
WHERE  s.lName = 'White' AND s.fName = 'John';
```

```
CREATE TYPE NameType AS (
    fName       VARCHAR(15),
    lName       VARCHAR(15)
);

CREATE TYPE PersonType AS (
    fName       VARCHAR(15),
    lName       VARCHAR(15),
                …
CREATE TYPE StaffType
  UNDER PersonType AS (
    staffNo  VARCHAR(5),
                …
```

Example

```
SELECT n.fName, n.lName
FROM   Staff s, UNNEST (s.relatives) AS n(fName, lName)
WHERE  s.lName = 'White' AND s.fName = 'John';
```

Staff

fName	lName	relatives		…
John	White	fName	lName	…
		Nancy	Allen	
		Cindy	Jones	
		Peter	Ward	

result of from clause

s.fName	s.lName	n.fName	n.lName
John	White	Nancy	Allen
John	White	Cindy	Jones
John	White	Peter	Ward

Collection Types

Operations on Multisets

- `MULTISET UNION` computes the union of two multisets:
  - adding the keyword `DISTINCT` removes duplicates
  - adding the keyword `ALL` retains duplicates

A
1
2
3
3
3
4
5

B
2
3
5

UNION DISTINCT
1
2
3
4
5

UNION ALL
1
2
2
3
3
3
3
4
5
5

Collection Types

Operations on Multisets

- `MULTISET INTERSECT` computes the intersection of two multisets:
  - the keyword `DISTINCT` removes duplicates
  - the keyword `ALL` places in the result as many instances of each value as the minimum number of instances of that value in either operand

A
1
2
3
3
3
4
5

B
2
2
3
3
4

INTERSECT DISTINCT
2
3
4

INTERSECT ALL
2
3
3
4

Collection Types

Operations on Multisets

- `MULTISET EXCEPT` computes the difference of two multisets:
  - the keyword `DISTINCT` removes duplicates
  - the keyword `ALL` places in the result a number of instances of a value, equal to the number of instances of the value in the first operand minus the number of instances of the second operand.

A
1
2
3
3
3
3
4
5

B
2
2
3
5
6

EXCEPT DISTINCT
1
3
4

EXCEPT ALL
1
3
3
3
4

Example

Queries that reference columns of a collection type

```
SELECT DeptNo, COUNT(Employees) AS NumberOfEmployees
FROM EMP
```

EMP	
DeptNo	Employees
10	{Clark, King, Miller}
20	{Smith, Ford, Adams, Scott, Jones}
30	{Allen, Blake, Martin, Turner, James, Ward}

result table	
DeptNo	NumberOfEmployees
10	3
20	5
30	6

Example

Queries that reference columns of a collection type

Query: In which department is „Turner“ working?

```
SELECT DeptNo
FROM EMP
WHERE 'Turner' IN Employees
```

EMP	
DeptNo	Employees
10	{Clark, King, Miller}
20	{Smith, Ford, Adams, Scott, Jones}
30	{Allen, Blake, Martin, Turner, James, Ward}

result table
DeptNo
30

Collection Types

EMPLOYEES	
DeptNo	lName
20	Smith
30	Allen
30	Ward
20	Jones
30	Martin
30	Blake
10	Clark
20	Scott
10	King
30	Turner
20	Adams
30	James
20	Ford
10	Miller

New Aggregate Functions for Multisets

- `COLLECT` creates a multiset from the value of the argument in each row of a group

Example

```
SELECT DeptNo, COLLECT(lName) AS Employees
FROM EMPLOYEES
GROUP BY DeptNo
```

Result table	
DeptNo	Employees
10	{Clark, King, Miller}
20	{Smith, Ford, Adams, Scott, Jones}
30	{Allen, Blake, Martin, Turner, James, Ward}

Collection Types

New Aggregate Functions for Multisets

- `FUSION` creates a multiset union of a multiset value in all rows of a group
- `INTERSECTION` creates the multiset intersection of a multiset value in all rows of a group

Predicates for use with Multisets

- comparison predicate (equality and inequality only)
- `DISTINCT` predicate
- `MEMBER` predicate
- `SUBMULTISET` predicate, which tests whether one multiset is a submultiset of another
- `IS A SET` / `IS NOT A SET` predicate, which checks whether a multiset is a set

Example

Use of the FUSION and INTERSECTION aggregate functions

The table *PropertyViewDates* describes the dates properties have been viewed by potential renters:

PropertyViewDates	
propertyNo	**viewDates**
PA14	MULTISET['14-May-16','24-May-16']
PG4	MULTISET['20-Apr-16,'14-May-16','26-May-16']
PG36	MULTISET['28-Apr-16','14-May-16']
PL94	Null

```
SELECT FUSION(viewDates) AS viewDateFusion,
 INTERSECTION(viewDates) AS viewDateIntersection
FROM PropertyViewDates;
```

produces the following result set:

result of query	
viewDateFusion	**viewDateIntersection**
MULTISET['14-May-16','24-May-16', '20-Apr-16','14-May-16,'26-May-16', '28-Apr16','14-May-16']	MULTISET['14-May-16']

Extensions of ORDBMSs

Large Object

is a data type that holds a large amount of data, such as a long text file or a graphics file:

- Binary Large Object (BLOB), a binary string that does not have a character set or collation association
- Character Large Object (CLOB)

In some database systems the BLOB is a noninterpreted byte stream, and the DBMS does not have any knowledge concerning the content of the BLOB or its internal structure.

This prevents the DBMS from performing queries and operations on inherently rich and structured data types, such as images, video, word processing documents, or Web pages.

In contrast, SQL large objects do allow some operations to be carried out in the DBMS server.

Large Objects

CLOB

The standard string operators can be used:

- concatenation operator (`string1 || string2`)
  returns the character string formed by joining the character string operands in the specified order

- character substring function `SUBSTRING(string FROM startpos FOR length)`
  returns a string extracted from a specified string from a start position for a given length

- character overlay function
  `OVERLAY(string1 PLACING string2 FROM startpos FOR length)`
  replaces a substring of `string1`, specified as a starting position and a length, with `string2`

Large Objects

CLOB

The standard string operators can be used:

- fold functions `UPPER(string)` and `LOWER(string)`
  convert all characters in a string to upper/lower case

- trim function `TRIM([LEADING | TRAILING | BOTH string1 FROM] string2)`
  returns `string2` with leading and/or trailing `string1` characters removed. If the `FROM` clause is not specified, all leading and trailing spaces are removed from `string2`

- length function `CHAR_LENGTH(string)`
  returns the length of the specified string

- position function `POSITION(string1 IN string2)`
  returns the start position of `string1` within `string2`

Large Objects

CLOB strings are not allowed to participate in most comparison operations, although they can participate in a LIKE predicate, and a comparison or quantified comparison predicate that uses the equals (=) or not equals (<>) operators.

As a result of these restrictions, a column that has been defined as a CLOB string cannot be referenced in

- a `GROUP BY` clause
- an `ORDER BY` clause
- a unique or referential constraint definition
- a join column
- in one of the set operations (`UNION`, `INTERSECT`, `EXCEPT`)

Large Objects

BLOB

A binary large object string is defined as a sequence of octets. All BLOB strings are comparable by comparing octets with the same ordinal position.

The following operators operate on BLOB strings and return BLOB strings, and have similar functionality as those defined previously:

- the BLOB concatenation operator (||)
- the BLOB substring function (`SUBSTRING`)
- the BLOB overlay function (`OVERLAY`)
- the BLOB trim function (`TRIM`)

In addition, the `BLOB_LENGTH` and `POSITION` functions and the `LIKE` predicate can also be used with BLOB strings.

Example

Use of Character and Binary Large Objects

Extend the Staff table to hold a resumé and picture for the staff member.

```
ALTER TABLE Staff
ADD COLUMN resume CLOB(50K);

ALTER TABLE Staff
ADD COLUMN picture BLOB(12M);
```

Advantages of ORDBMSs

Weaknesses of RDBMSs

ORDBMSs resolve many of the mentioned weaknesses of conventional RDBMS.

Reuse and Sharing

standard functionality can be performed centrally, rather than have it coded in each application

Preserving Knowledge, Experience, and Investments

ORDBMs should allow organizations to take advantage of the new extensions in an evolutionary way without losing the benefits of current database features and functions.

SQL:2011 standardizes many object-relational and other extensions

SQL:2011 is designed to be compatible with SQL2 standard of RDBMs

Disadvantages of ORDBMS

- complexity
  - the simplicity and purity of the relational model are lost with most of object-relational extensions
- increased costs
- many extensions serve only a minority of applications
- object-relational extensions don't adequately represent object-oriented models
  - in most cases, object applications are not as data-centric as relational-based ones
  - objects are fundamentally not extensions of data, but a completely different concept with far greater power to express real-world relationships and behaviors