SGG 4653
Advance Database System

Object-Relational DBMS

Outline

- Advantages and disadvantages of ORDBMS
- ORDBMS Features
- SQL3 – New OO Data Management Features

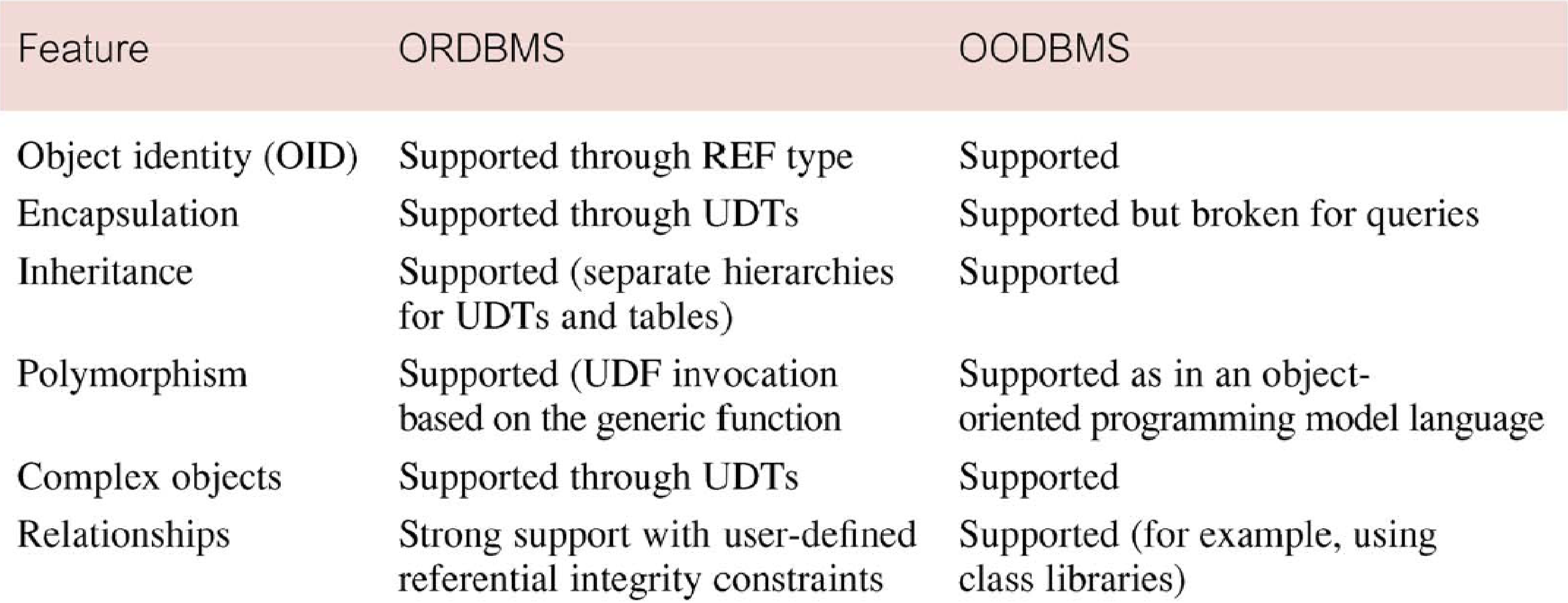
Advantages of ORDBMS

- Resolves many of known weaknesses of RDBMS.
- Reuse and sharing:
  - reuse comes from ability to extend server to perform standard functionality centrally
  - gives rise to increased productivity both for developer and end-user.
- Preserves significant body of knowledge and experience gone into developing relational applications.

Disadvantages of ORDBMS

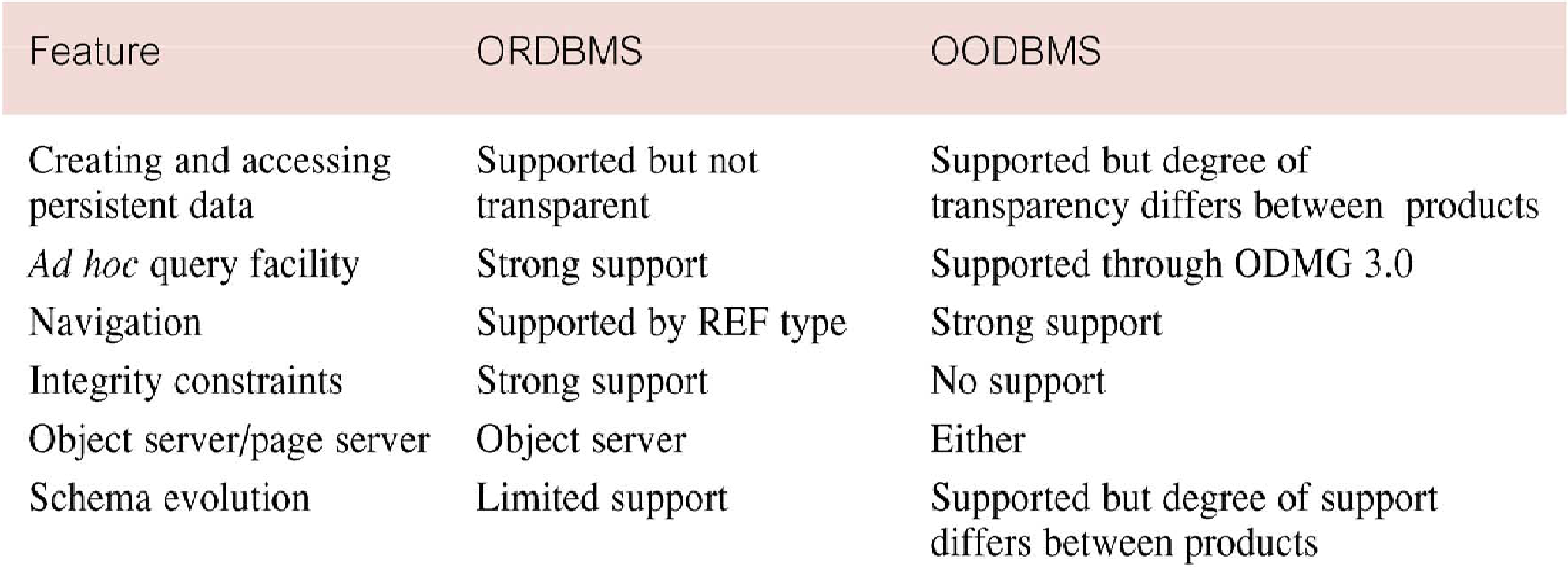
- Complexity.
- Increased costs.
- Proponents of relational approach believe simplicity and purity of relational model are lost.
- Some believe RDBMS is being extended for what will be a minority of applications.
- SQL now extremely complex.

Data Modeling Comparison of OR & OO DBMS



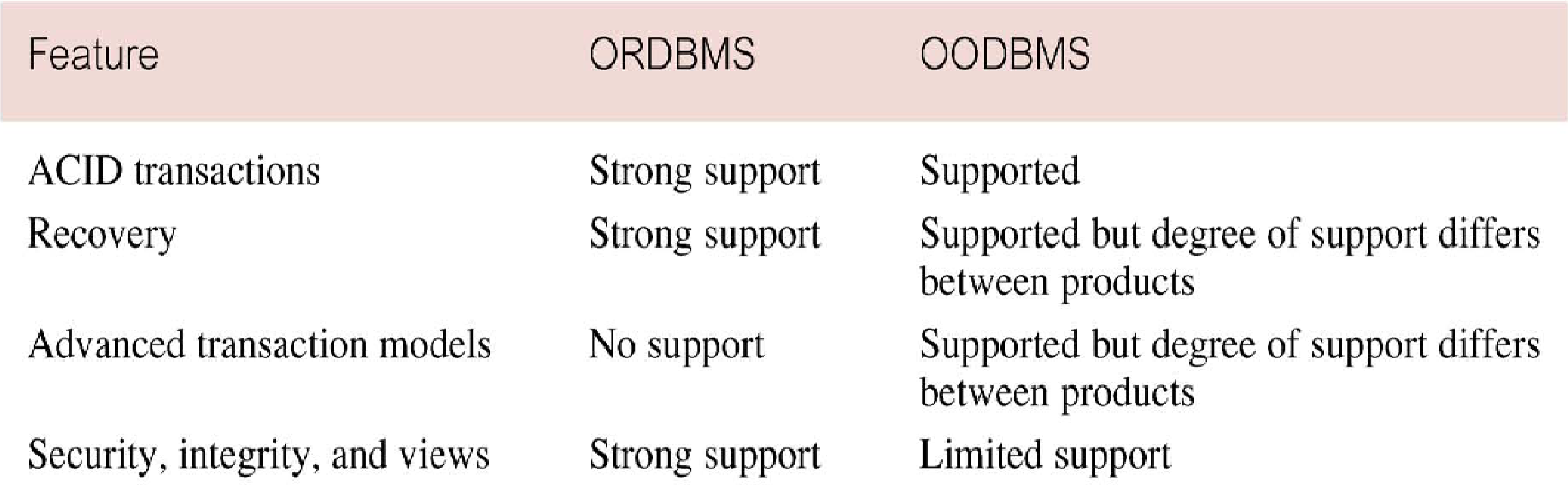
Feature	ORDBMS	OODBMS
Object identity (OID)	Supported through REF type	Supported
Encapsulation	Supported through UDTs	Supported but broken for queries
Inheritance	Supported (separate hierarchies for UDTs and tables)	Supported
Polymorphism	Supported (UDF invocation based on the generic function	Supported as in an object-oriented programming model language
Complex objects	Supported through UDTs	Supported
Relationships	Strong support with user-defined referential integrity constraints	Supported (for example, using class libraries)

Data Access Comparison of OR & OO DBMS



Feature	ORDBMS	OODBMS
Creating and accessing persistent data	Supported but not transparent	Supported but degree of transparency differs between products
Ad hoc query facility	Strong support	Supported through ODMG 3.0
Navigation	Supported by REF type	Strong support
Integrity constraints	Strong support	No support
Object server/page server	Object server	Either
Schema evolution	Limited support	Supported but degree of support differs between products

Data Sharing Comparison of OR & OO DBMS



Feature	ORDBMS	OODBMS
ACID transactions	Strong support	Supported
Recovery	Strong support	Supported but degree of support differs between products
Advanced transaction models	No support	Supported but degree of support differs between products
Security, integrity, and views	Strong support	Limited support

A- Atomicity
C – Consistent
I – Isolation
D - Durability

ORDBMS Features

OO features being added include:

- User-extensible types
- Encapsulation
- Inheritance
- Polymorphism
- Dynamic binding of methods
- Complex objects including non-1NF objects
- Object identity

SQL3 – New OO Data Management Features

- Type constructors for row types and reference types.
- User-defined types (distinct types and structured types) that can participate in supertype / subtype relationships.
- User-defined procedures, functions, and operators.
- Type constructors for collection types (arrays, sets, lists, and multisets).
- Support for large objects–Binary Large Object (BLOBs) and Character Large Object (CLOBs).
- Recursion.

SQL3 – New OO Data Management Features

Row Types

- Sequence of field name/data type pairs that provides data type to represent types of rows in tables.
- Allows complete rows to be:
  - stored in variables,
  - passed as arguments to routines,
  - returned as return values from function calls.
- Also allows column of table to contain row values.

SQL3 – New OO Data Management Features

Example 1 – Use of Row Types

```
CREATE TABLE Branch (branchNo CHAR(4),
    address ROW(street  VARCHAR(25),
                 city       VARCHAR(15),
                 postcode ROW(cityIdentifier VARCHAR(4),
                                      subPart VARCHAR(4))));
```

```
INSERT INTO Branch
VALUES (‘B005’, (’22 Deer Rd’, ‘London’,
                          ROW(‘SW1’, ‘4EH’)));
```

SQL3 – New OO Data Management Features

Named Row Type

- A named row type is a row type with a name assigned to it.
- A named row type is effectively a user defined data type with a non-encapsulated internal structure (consisting of its fields).

SQL3 – New OO Data Management Features

Example 2 – Use of Named Row Type

```
CREATE ROW TYPE account_t
    (acctno INT,
     cust REF(customer_t),
     type CHAR(1),
     opened DATE,
     rate DOUBLE PRECISION,
     balance DOUBLE PRECISION,
    );
```

```
CREATE TABLE account OF account_t
    (PRIMARY KEY acctno
    );
```

SQL3 – New OO Data Management Features

User-Defined Types (UDTs)

- Subdivided into two categories: distinct types and structured types.
- Distinct type allows differentiation between same underlying base types:

```
CREATE TYPE OwnerNoType AS VARCHAR(5) FINAL;

CREATE TYPE StaffNoType AS VARCHAR(5) FINAL;
```

FINAL – indicates that we cannot create subtypes of this user-defined type

SQL3 – New OO Data Management Features

User-Defined Types (UDTs)

§ Value of an attribute can be accessed using common dot notation:

(assuming p is an instance of the UDT PersonType which has an attribute fName of type VARCHAR. We can access fname attribute as:

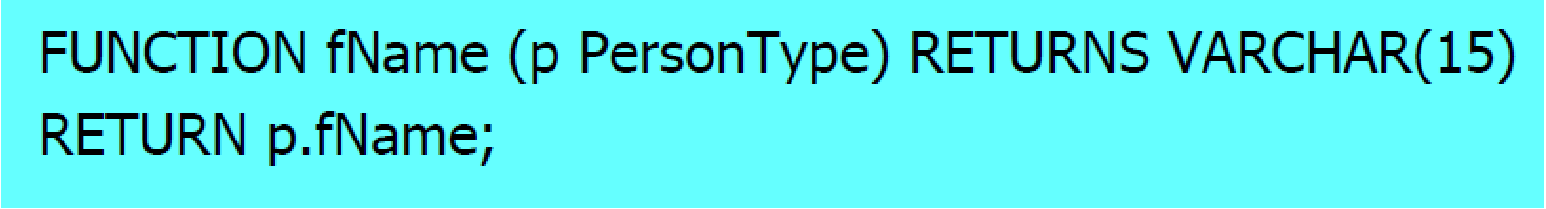
```
p.fName
```

```
p.fName = ‘A. Smith’
```

SQL3 – New OO Data Management Features

User-Defined Types (UDTs)

- For each attribute, an observer (get) and a mutator (set) function are automatically defined, but can be redefined by user in UDT definition.
- the observer (get) function for the fName attribute of PersonType:

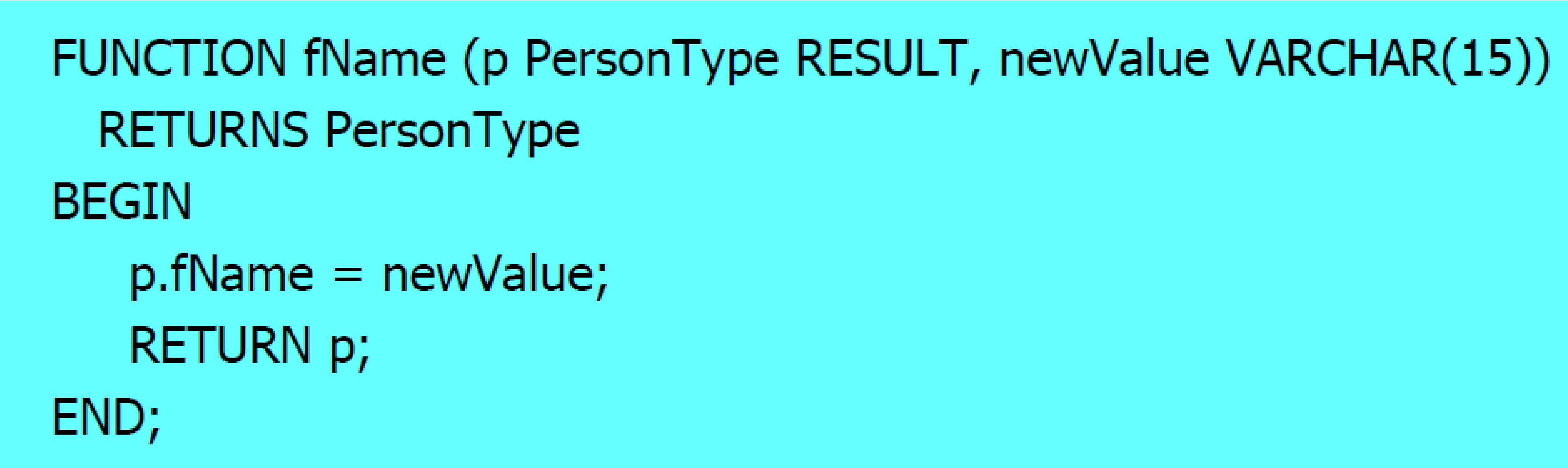


```
FUNCTION fName (p PersonType) RETURNS VARCHAR(15)
RETURN p.fName;
```

SQL3 – New OO Data Management Features

User-Defined Types (UDTs)

- The mutator (set) function to set the value to newValue is:



```
FUNCTION fName (p PersonType RESULT, newValue VARCHAR(15))
   RETURNS PersonType
BEGIN
     p.fName = newValue;
     RETURN p;
END;
```

SQL3 – New OO Data Management Features

```
CREATE TYPE employee_t
( PUBLIC
    name CHAR(20),
    b_address address_t,
    manager employee_t,
    hiredate DATE,
  PRIVATE
    base_salary DECIMAL(7,2),
    commission DECIMAL(7,2),
PUBLIC
    FUNCTION working_years (p
    employee_t) RETURNS
    INTEGER
    <code to calculate number of
    working years>,
```

```
PUBLIC
    FUNCTION working_years (p
    employee_t, y years) RETURNS
    employee_t
    <code to update number of working
    years>,
PUBLIC
    FUNCTION salary (p employee_t)
    RETURNS DECIMAL
    <code to calculate salary>
);
```

SQL3 – Definition of new UDT

```
CREATE TYPE PersonType AS (
        dateOfBirth DATE CHECK (dateOfBirth >
                                         DATE '1900-01-01'),
        fName    VARCHAR(15)      NOT NULL,
        lName    VARCHAR(15)      NOT NULL,
        sex                 CHAR,
        FUNCTION age (p PersonType) RETURNS INTEGER
         RETURN  code to get age from dateOfBirth
        END,
        FUNCTION age (p PersonType RESULT,
                   DOB: DATE) RETURNS PersonType
        RETURN  set dateOfBirth
       END)
    REF IS SYSTEM GENERATED
    INSTANTIABLE
    NOT FINAL;
```

REF IS SYSTEM GENERATED – indicates that the actual values of the associated REF type are provided by the system

INSTANTIABLE – indicates that instances can be created for this type

NOT FINAL – indicates that we can create subtypes of this user-defined type

SQL3 – Definition of new UDT

```
CREATE TYPE StaffType UNDER PersonType AS (
    staffNo          VARCHAR(5)      NOT NULL          UNIQUE,
    position         VARCHAR(10)     DEFAULT ` Assistant’,
    salary           DECIMAL(7, 2),
    branchNo         CHAR(4),
   CREATE FUNCTION isManager (s StaffType) RETURNS BOOLEAN
   BEGIN
     IF s.position = ‘Manager’ THEN
          RETURN TRUE;
     ELSE
          RETURN FALSE;
     END IF
END)
INSTANTIABLE
NOT FINAL;
```

INSTANTIABLE – indicates that instances can be created for this type.

NOT FINAL – indicates that we can create subtypes of this user-defined type

SQL3 – Table Creation using UDT

```
CREATE TABLE Staff (
    info                StaffType,
    PRIMARY KEY         (staffNo));
```

or

```
CREATE TABLE Staff OF StaffType (
    REF IS  staffID  SYSTEM GENERATED,
    PRIMARY KEY         (staffNo));
```

Indicates that the actual values of associated REF are provided by the system

SQL3 – Using Reference Type to Define a Relationship

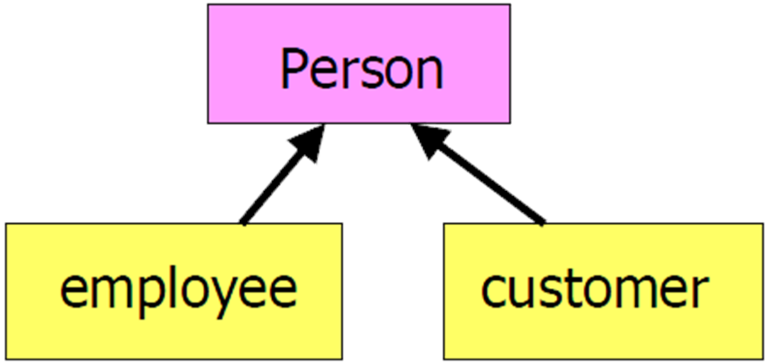
```
CREATE TABLE PropertyForRent (
   propertyNo         PropertyNumber          NOT NULL,
   street             Street   NOT NULL,
    city              City     NOT NULL,
    postcode         PostCode,
    type        PropertyType    NOT NULL DEFAULT ‘F’,
    rooms       PropertyRooms  NOT NULL DEFAULT 4,
    rent        PropertyRent     NOT NULL DEFAULT 600,
     staffID     REF(StaffType)          SCOPE   Staff
              REFERENCES   ARE   CHECKED   ON   DELETE
   CASCADE,
     PRIMARY KEY (propertyNo));
```

SCOPE specifies the associated referenced table

REFERENCES ARE CHECKED indicates the referential integrity is to be maintained

- This example used a reference type, REF(StaffType) to model the relationship between PropertyForRent and Staff

SQL3 – Creation of Subtable (Inheritance)



```
CREATE TABLE person (name CHAR(20), sex CHAR (1), age
    INTEGER);

CREATE TABLE employee UNDER person (salary FLOAT);

CREATE TABLE customer UNDER person (account INTEGER);
```

SQL3 – Creation of Subtable (Inheritance)

```
CREATE TABLE Manager UNDER Staff (
        bonus                   DECIMAL(5, 2),
        mgrStartDate            DATE);
```

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

SQL3 – Use of UDFs

- Example: List flats that are for rent at branch B003.
- We might decide to use a function:

```
CREATE FUNCTION flatTypes()
                    RETURNS SET(PropertyForRent)
      SELECT * FROM PropertyForRent

      WHERE type = ‘Flat’;
```

And the query become:

```
SELECT propertyNo, street, city, postcode
  FROM TABLE (flatTypes())
  WHERE branchNo = ‘B003’;
```

SQL3 – Use of UDFs

- Query Processer should ‘flatten’ that query using the following step:

(1)

```
SELECT propertyNo, street, city, postcode
FROM TABLE (SELECT * FROM   PropertyForRent
  WHERE type = ‘Flat’)
WHERE branchNo = ‘B003’;
```

(2)

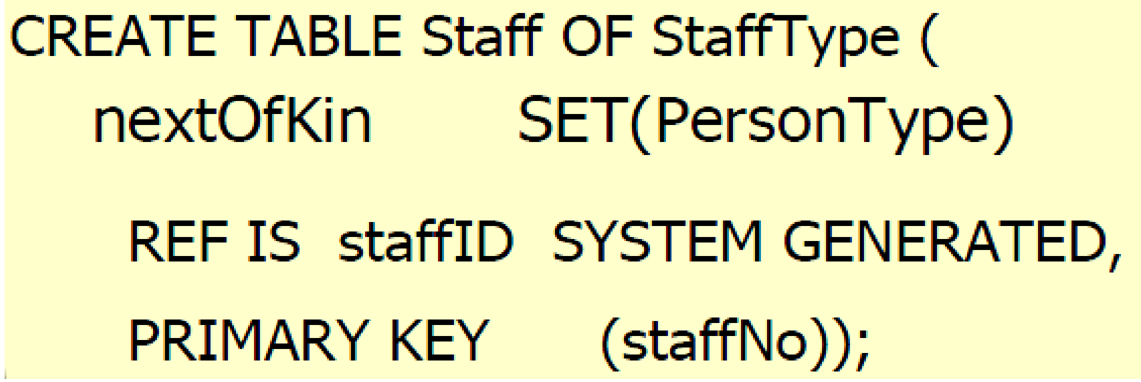
```
SELECT propertyNo, street, city, postcode
FROM PropertyForRent
WHERE type = ‘Flat’ AND branchNo = ‘B003’;
```

SQL3 – Collection Types

- ARRAY: 1D array with maximum number of elements.
- LIST: ordered collection that allows duplicates.
- SET: unordered collection that does not allow duplicates.
- MULTISET: unordered collection that allows duplicates.

SQL3 – Use of collection SET

- Extend Staff table to contain details of a number of next of kin, and then: Find first and last names of John White’s next-of-kin.
- We could implement the column as an ARRAY data type:

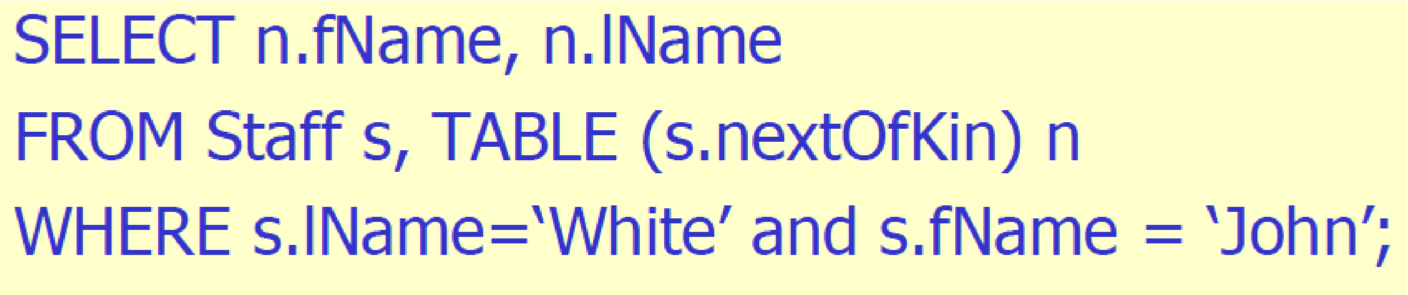


```
CREATE TABLE Staff OF StaffType (
   nextOfKin       SET(PersonType)

     REF IS  staffID  SYSTEM GENERATED,

     PRIMARY KEY      (staffNo));
```

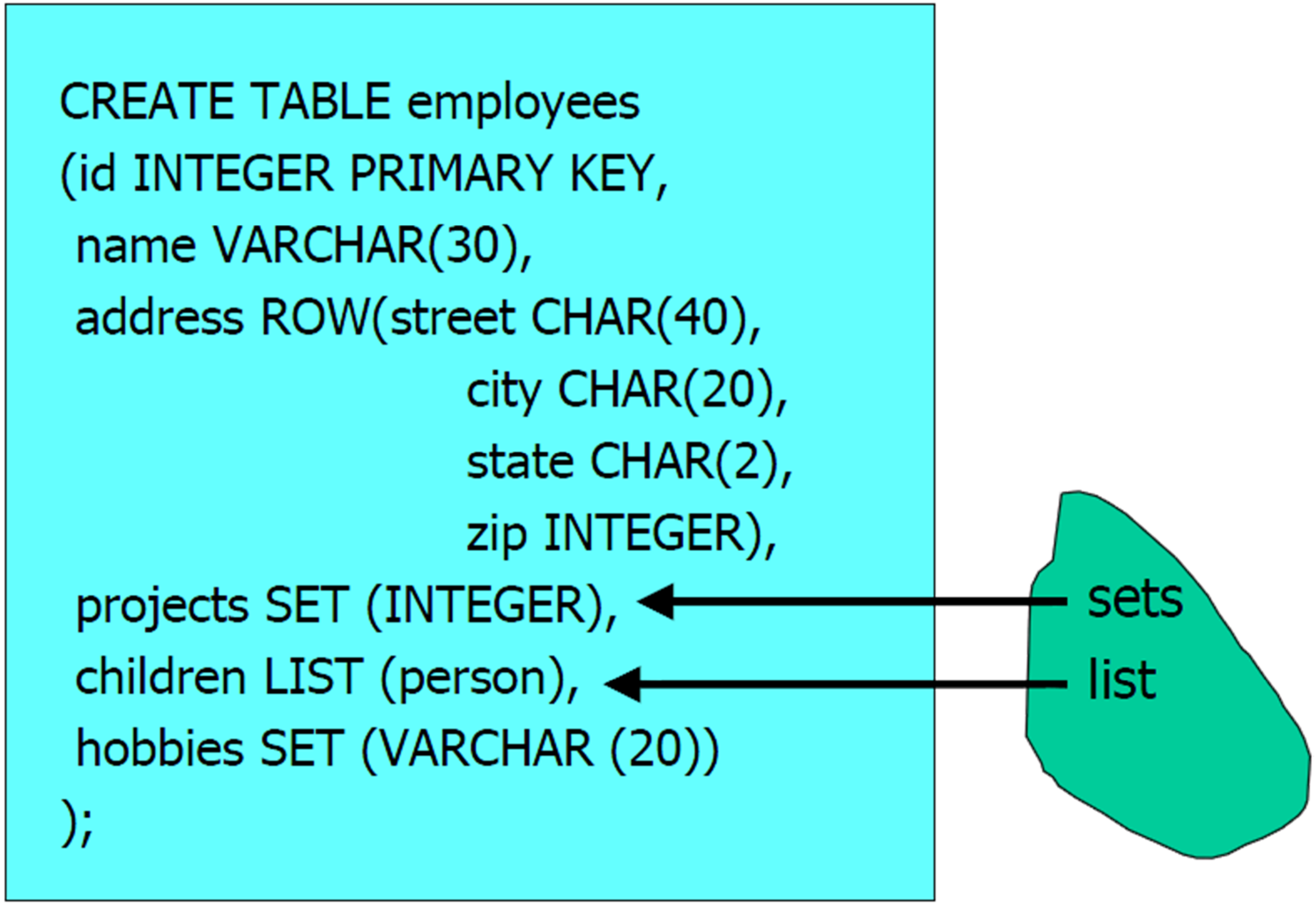
Query becomes:



```
SELECT n.fName, n.lName
FROM Staff s, TABLE (s.nextOfKin) n
WHERE s.lName=‘White’ and s.fName = ‘John’;
```

SQL3 – Collection Types

- Example: Defines Collection types for sets, and lists.



```
CREATE TABLE employees
(id INTEGER PRIMARY KEY,
 name VARCHAR(30),
 address ROW(street CHAR(40),
                        city CHAR(20),
                        state CHAR(2),
                        zip INTEGER),
 projects SET (INTEGER),
 children LIST (person),
 hobbies SET (VARCHAR (20))
);
```

SQL3 – Retrieve Specific Column/Rows

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

- Find the names of all Managers.
- Uses implicitly-defined observer (get) function position.

SQL3 – Retrieve specific components of a row type

Row types define types for tuples, and they can be nested.

```
CREATE ROW TYPE AddressType{
        street   CHAR(50),
        city     CHAR(25),
        zipcode  CHAR(10)
   }
```

```
CREATE ROW TYPE  PersonType{
        name    CHAR(30),
        address  AddressType,
        phone    phoneNumberType
}
```

```
CREATE TABLE Person OF TYPE
    PersonType;
```

Recall: row types can be nested!

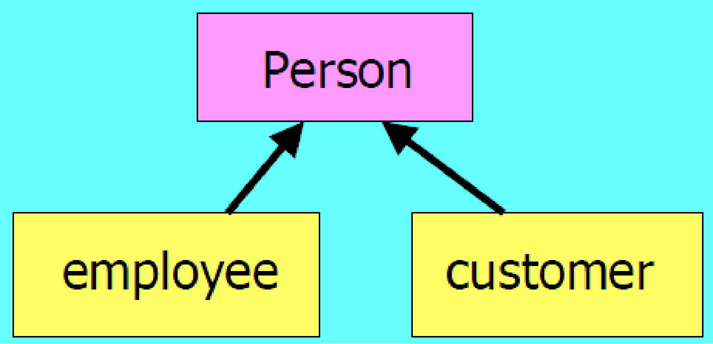
Accessing components of a row type: (double dots)

```
SELECT  Person.name,
Person.address..city

FROM     Person

WHERE  Person.address..street
LIKE  '%Mountain%'
```

SQL3 – Use of ONLY



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

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

- This will list out not only records explicitly inserted into Person table, but also records inserted directly/indirect into subtables of Person.
- Can restrict access to specific instances of Person table, excluding any subtables, using ONLY.

SQL3 – Large Objects

- A table field that holds large amount of data.
- Three different types:
  - Binary Large Object (BLOB)
  - Character LOB (CLOB)
  - National CLOB
- In SQL3, LOB allows some operations to be carried out in DBMS server.

SQL3 – Use of CLOB and BLOB

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

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

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

```
CREATE TABLE employees
    (id INTEGER,
     name VARCHAR(30),
     salary us_dollar,
     ...
     resume CLOB(75K),
     signature BLOB(1M),
     picture BLOB(12M)
    );
```

SQL3 – Recursion

- Linear recursion is major new operation in SQL3.

```
WITH RECURSIVE
 AllManagers (staffNo, managerStaffNo) AS
    (SELECT staffNo, managerStaffNo
     FROM Staff
    UNION
    SELECT in.staffNo, out.managerStaffNo
    FROM AllManagers in, Staff out
    WHERE in.managerStaffNo = out.staffNo)
 SELECT * FROM AllManagers
 ORDER BY staffNo, managerStaffNo;
```

Staff

staffNo	managerstaffNo
S005	S004
S004	S003
S003	S002
S002	S001
S001	NULL