

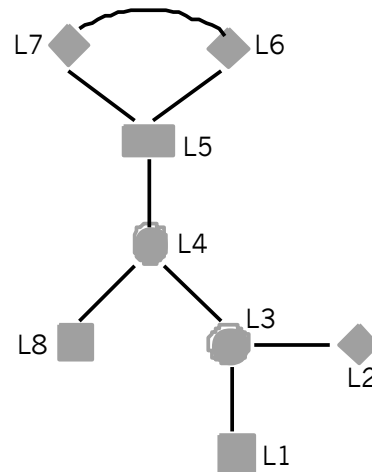
INTRODUCTION TO OBJECT-ORIENTED DATABASES

U08049 Database design - 2013





Introduction

- Differences between Object Oriented Databases and Relational Database.
- Principles of Object Orientation
- Data representations in OODB
- Designing OO databases
- Methods of Objects in Databases

Complex data



FIGURES

SHAPE	PRAMETERS	properties
	a
	a, angle
	a, b
	← r

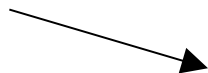
Representation

NETWORK









LNODE	RNODE	connect-details
L1	L3
L2	L3
L3	L4
L4	L5
L4	L8	
L5	L6
L5	L7
L6	L7
L8	L1

Flattened

Structure



NODES

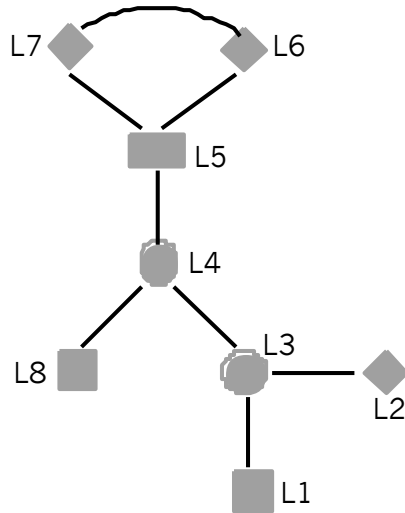
CODE	SHAPE	node-detail
L1	
L2	
L3	
L4	
L5	
L6	
L7	
L8	

*Table-specific
operations*





Relational Shortcomings

- Each domain must be *atomic*
- Anything but relational operations must be coded away from the data (as queries and DBMS operations)
- Inherently complex structures must be normalised (i.e. flattened)
- Coded operations are difficult to re-use, maintain etc.

Relaxation of the Relational Model



FIGURES

SHAPE	PARAMETERS	properties
	a
	a, angle
	a, b
	r









Shapes - parameters and operations on shapes

Nodes - including links to and from, and operations on nodes and their links

NETWORK

LNODE	RNODE	connect-details
L1	L3
L2	L3
L3	L4
L4	L5
L4	L8
L5	L6
L5	L7
L6	L7
L8	L1

NODES

CODE	SHAPE	node-detail
L1	
L2	
L3	
L4	
L5	
L6	
L7	
L8	

Networks - (if required)

Minimal Relaxation

- Definition for new object structures:
- *An object type is a subset of an expanded cartesian product of n , not necessarily distinct domains $D_1 \times D_2 \times \dots \times D_n$, such that for every element $d^k = \langle d^k_1, d^k_2, \dots, d^k_n \rangle \in R$ a predefined proposition $p(\langle d^k_1, d^k_2, \dots, d^k_n \rangle)$ is true; $d^k_i \in D_i$ for every $i=1, 2, \dots, n$*
- Aim is to enhance the concept of domains to allow more complex data in the database

Closure

- Any (legal) application of relational operators (select, project, join) on a set of relations produces a relation.

Implications

- Domains may now be defined as complex objects
- Property of 'closure' is now meaningless, and unenforceable anyway (why?)
- New structures should contain operations as well as data

OO Database Shortcoming

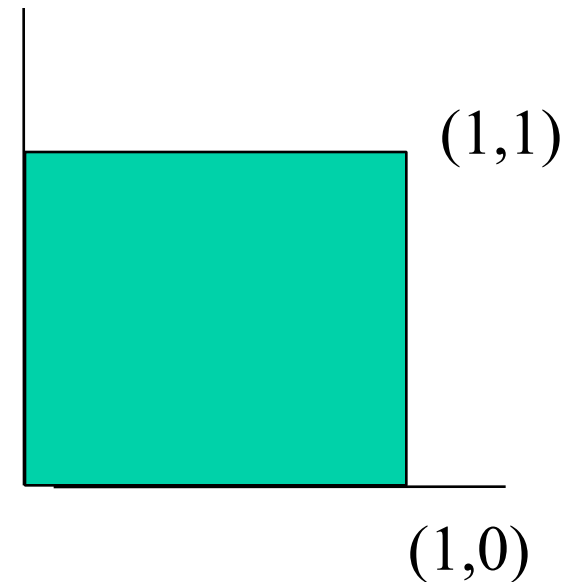
- It is very difficult to match the efficiencies of the relational model
- This also has consequences for processing speed
- Relaxation of the relation concept may imply unclear relations
- Object Oriented concepts may introduce unintended relations (e.g. via inheritance or polymorphism)

Object-Orientation

- Object classes and instances
- Encapsulation
- Inheritance
- Polymorphism

Object - Example 1

- The rectangle R1 $\{(0,0), (1,0), (0,1), (1,1)\}$ is an object
- It belongs to the class RECTANGLE
- *return_area* and *enlarge* are possible methods
- `return_area(R1)` and `enlarge(R1, 2)` are associated messages.



Object - Example 2

- The student P99456312, CO/IF, {8046, 8005, 8976}, 23 is an object
- It belongs to the class STUDENT
- *graduate* and *resit* are possible methods
- `graduate(P99456312, BSc, Ili)` and `resit(P99456312, 8046)` are associated messages



Encapsulation

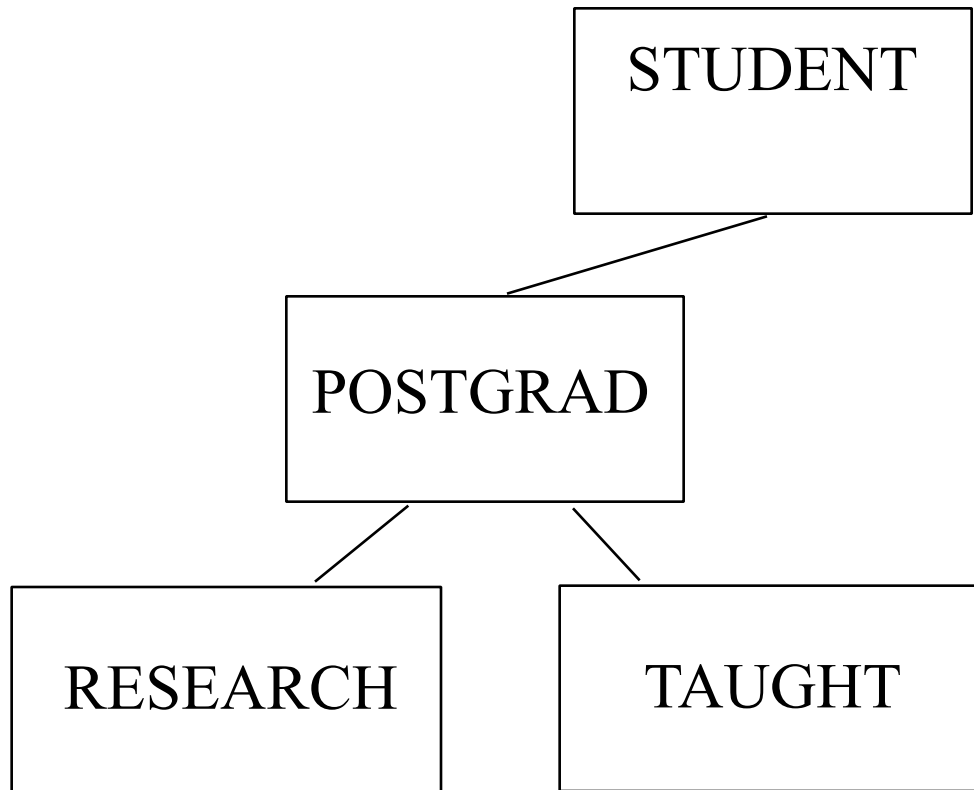
- Private memory
 - a set of attributes representing the internal state of the object
 - accessible through the methods of the object
- Public interface
 - interface definitions for the methods of the object
 - pre-defined function calls to the object

Implications

- Nothing can alter the private memory of an object instance except the execution of a method
- Methods can only be defined in terms of the changes they make to object instances

Class Hierarchies and Inheritance

Structural and behavioural inheritance



method : `get_result`
attributes : `name, number`

Methods and Inheritance

- In the previous example, the method `get_result`, and the attributes `name` and `number` are inherited by subclasses
- The actual code to be executed can be overwritten by the methods of the subclasses
- Details of methods (and attributes) stored in a 'type defining object' – see later

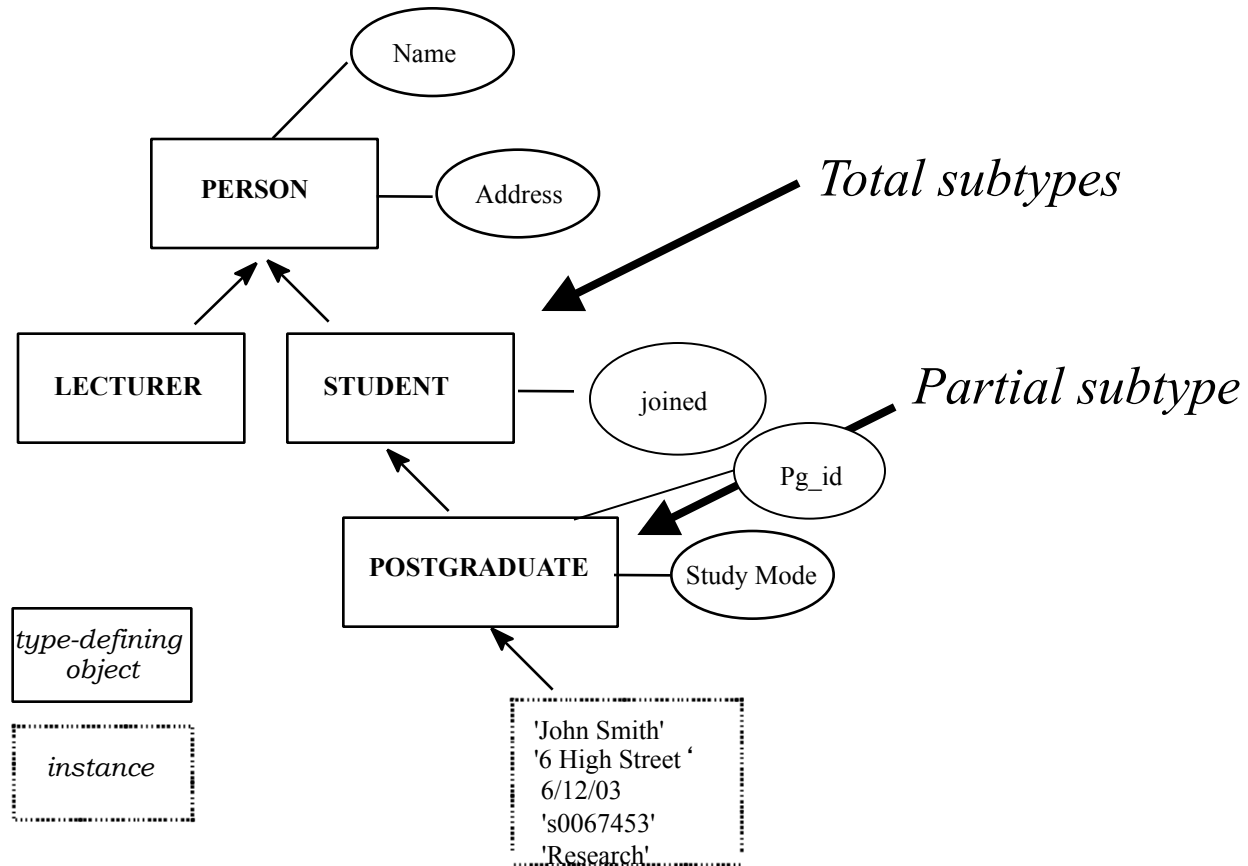
Polymorphism

- The method 'get_result' in the previous example may be redefined for each subclass
- The ability to apply the same method (name) to different classes is known as polymorphism
- e.g. overloaded addition:
 $1 + 4 = 5$
 "a" + "b" = "ab"

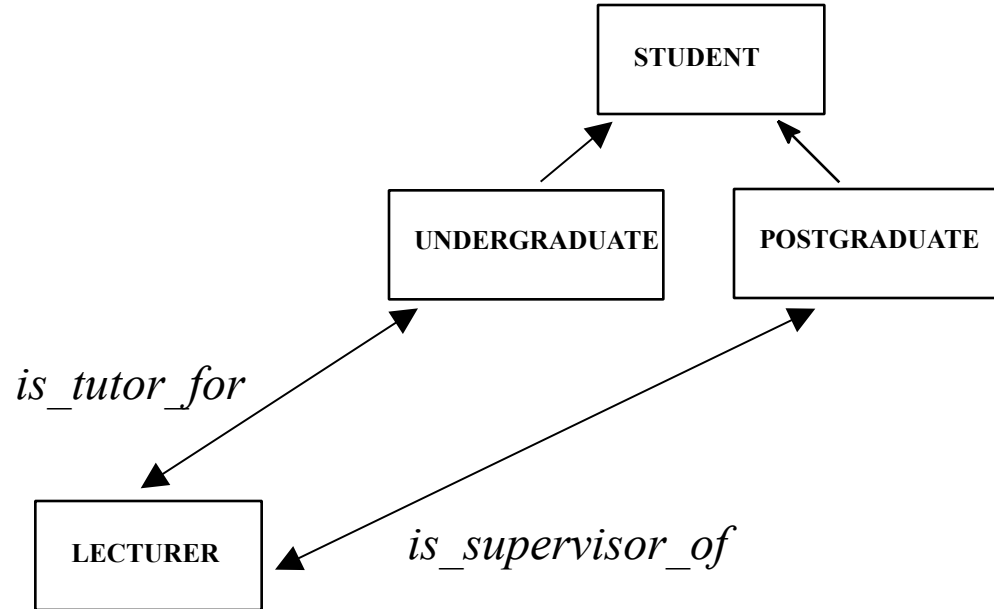
Databases

- Store persistent data, independently from applications
- Defined using a data definition language (DDL)
- Manipulated using a data manipulation language (DML)
- OO databases may include methods as part of the 'data' base

Type-defining Objects



Relationships are defined for Object Types



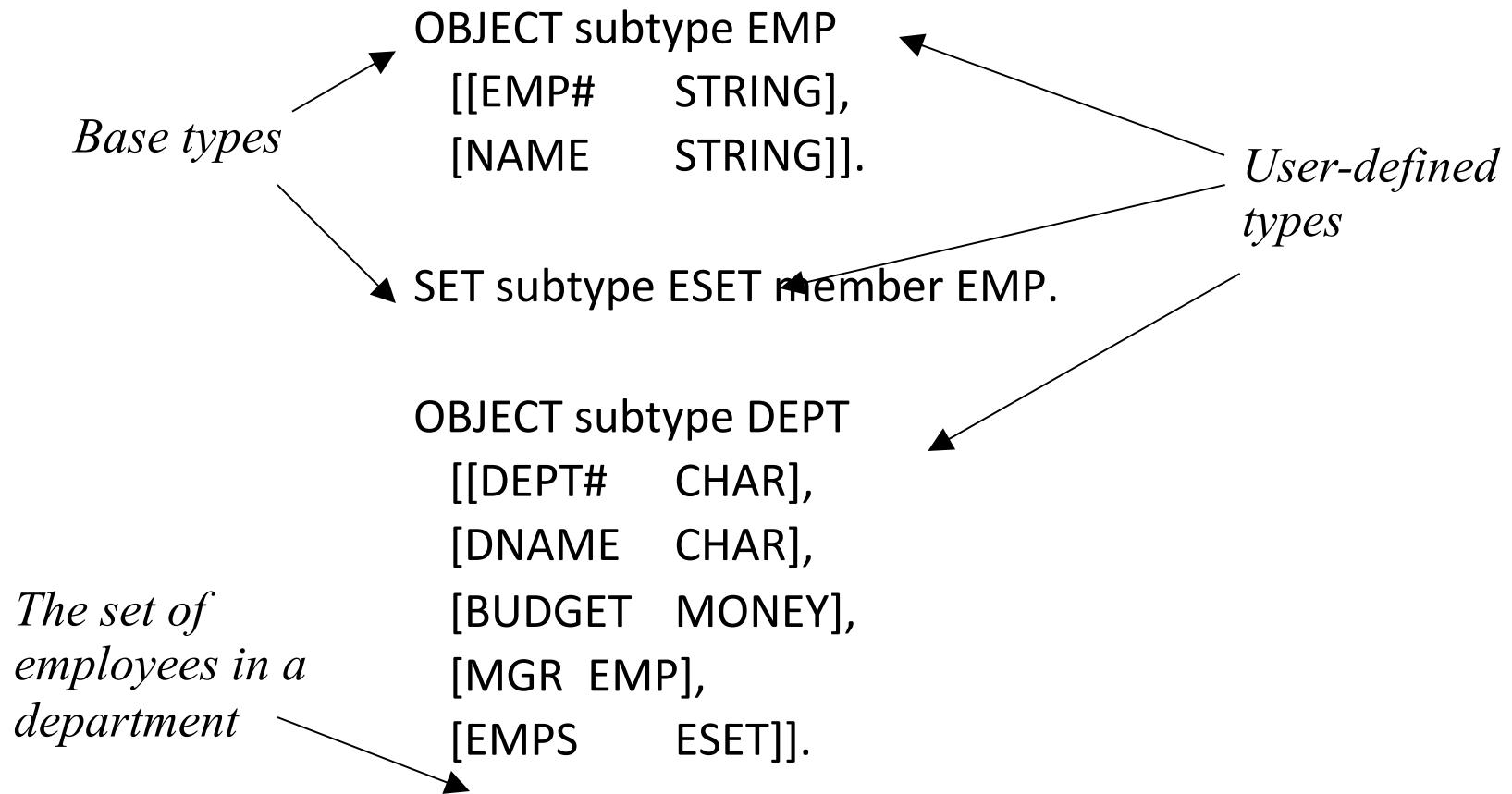
Data Definition

- Need to define :
 - object types, and any hierarchies
 - relationships which exist between types
 - instances of types
- For Relational DB – no object types, just tables

Introduction to the Notations

- ‘Pure’ object-oriented databases
 - for reference only;
 - illustrate principles
- Oracle Object-Relational databases
 - Industry standard
 - additions to relational model to allow some OO implementation
 - expensive

OO - Defining Types and Sets



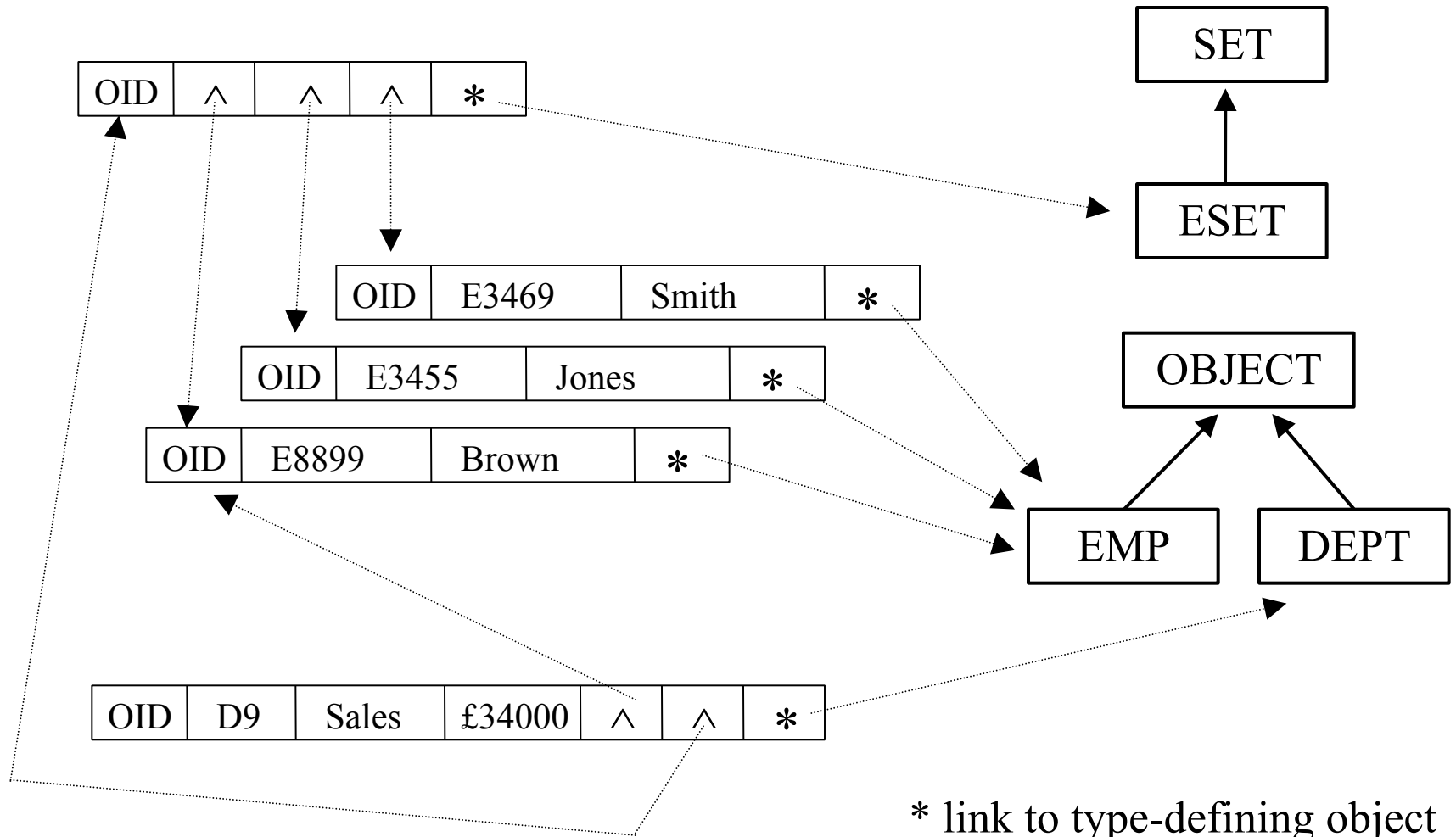
OO - Notes on the Class

- The attribute 'MGR' takes a value which 'points to' an object of the class 'EMP'
- The attribute 'EMPS' takes a value which 'points to' an object of type 'ESET'
- The particular SET pointed to by a particular instance of DEPT, contains pointers to the employees of that department.

Object Identifiers (OIDs)

- Each object (instance) is given a unique identifier by the DBMS
- The OID is used internally to reference an object instance (think: *like* a reference or pointer)
- Externally, at the user interface, the OID is not known

Object Types and Instances



Object-Relational Databases

- Oracle allows object types to be created, and subsequently tables of the defined objects
- Types and column types for inheritance and aggregation
- OIDs can be used to link object instances for associations

OBJECT RELATIONAL DATA

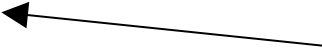
OR - Defining Object Types

```
CREATE TYPE PERSON_T AS OBJECT (  
  NAME  VARCHAR2(20),  
  ADDR  VARCHAR2(50));  
/
```

```
CREATE TYPE STUDENT_T AS OBJECT (  
  JOINED DATE,  
  PERSON    PERSON_T);  
/
```

```
CREATE TYPE PGRAD_T AS OBJECT(  
  PG_ID  VARCHAR2(9),  
  ST_MODE VARCHAR2(15),  
  STUDENT STUDENT_T);  
/
```

*No table - the
type forms part
of the hierarchy*



OR - Tables of Objects

- We can now define a PGRAD object table:

```
CREATE TABLE PGRAD OF PGRAD_T (  
PG_ID NOT NULL PRIMARY KEY);
```

- PGRAD_T is a nested structure (or complex object), as illustrated by the following operations
- It has 3 attributes

OR - Inserting Data

- Adding data to PGRAD:

insert into PGRAD values

```
( 's00091347' ,  
  'Research' ,  
  student_t(' 12-Jan-03' ,  
            person_t( 'Smith' , '43 High St, Withenspoon' )))
```

- The student_t structure is the third attribute of PGRAD

OR - Retrieving Data

- Retrieving data from PGRAD:

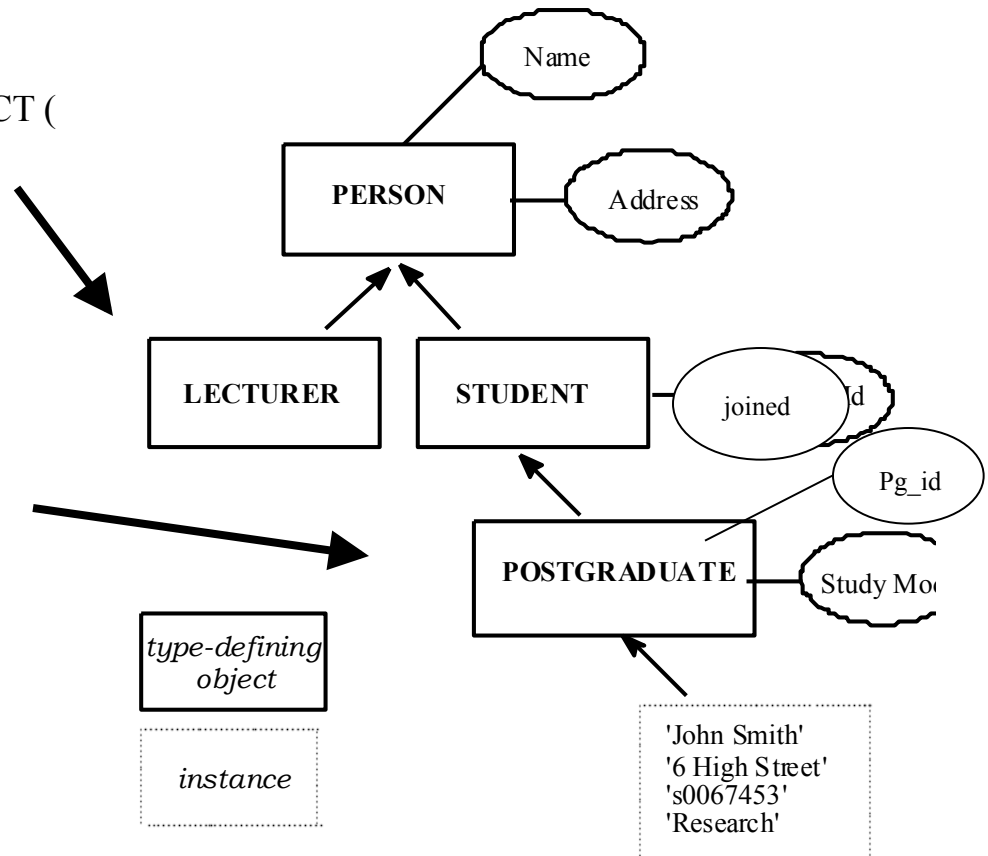
```
select p.pg_id, p.student.joined, p.student.person.name  
from pgrad p;
```

- p is an alias for pgrad – always use!
- The ‘dot’ notation uses the attribute names from the structure being accessed

OR - Implementing a Hierarchy

```
CREATE TYPE LECTURER_T AS OBJECT (  
  ST_ID    VARCHAR2(8),  
  PERSON  PERSON_T);  
/
```

```
CREATE TYPE PGRAD_T AS OBJECT (  
  PG_ID VARCHAR2(9),  
  ST_MODE VARCHAR2(15),  
  STUDENT STUDENT_T);  
/
```



OR - Important Note!

- In this hierarchy, STUDENT is not a 'leaf node'; therefore we do not create a table for STUDENT
- Create types for PERSON, LECTURER, STUDENT, POSTGRADUATE
- Create tables for (just the leaf nodes) LECTURER and POSTGRADUATE

OR - Notes on Primary Keys

- In Oracle, an attribute to be used as the primary for a table, **MUST** be declared in the type for which the table is declared.

```
CREATE TYPE PGRAD_T AS OBJECT (  
    PG_ID VARCHAR2(9),  
    ST_MODE VARCHAR2(15),  
    STUDENT STUDENT_T);  
  
CREATE TABLE PGRAD OF PGRAD_T (  
    PG_ID NOT NULL PRIMARY KEY);
```

Aggregation

“ The domain of an object type may be formed by aggregating domains of other object types. This means that the values and operations of the component domains are available to the new object type, as well as operations defined on the new domain.”

“ Aggregation is the process whereby a set of existing domains are combined to form a new domain. An instance of the type defined on this new domain comprises a set of values drawn from the component domains.”

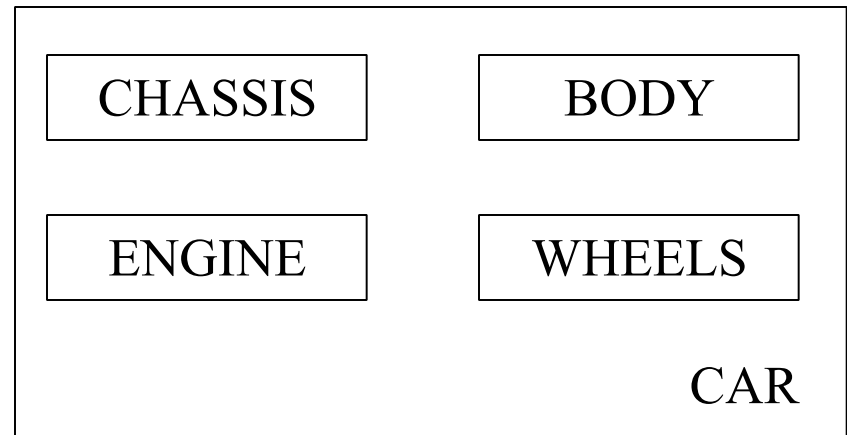
Aggregated Object Types

```
CREATE TYPE CHASSIS_T AS OBJECT(  
    CH_NUM  VARCHAR2(15),  
    MAKE    VARCHAR2(20));  
/
```

```
CREATE TYPE BODY_T AS OBJECT(  
    COLOUR  VARCHAR2(12),  
    STYLE   VARCHAR2(10));  
/
```

```
CREATE TYPE ENGINE_T AS OBJECT(  
    CC  INTEGER);  
/
```

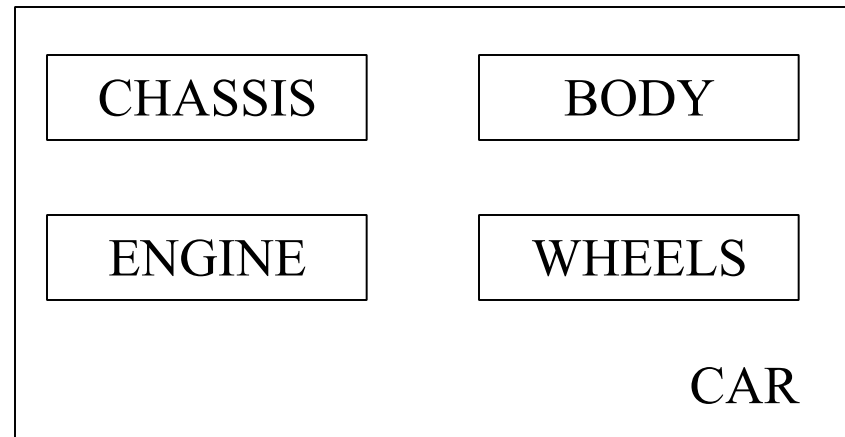
```
CREATE TYPE WHEELS_T AS OBJECT(  
    AMOUNT  INTEGER,  
    DESTION VARCHAR2(10));  
/
```



```
CREATE TYPE CAR_T AS OBJECT(  
    CHASSIS  CHASSIS_T,  
    BODY     BODY_T,  
    ENGINE   ENGINE_T,  
    WHEELS   WHEELS_T);  
/
```

```
CREATE TABLE CAR OF CAR_T;
```

Inserting and Retrieving Data with Aggregated Types



```
INSERT INTO CAR VALUES(  
    CHASSIS_T( 'CTW12543' , 'FORD' ),  
    BODY_T( 'RACING GREEN' , 'SALOON' ),  
    ENGINE_T(1796),  
    WHEELS_T(3, 'SPORTS' ));
```

```
SELECT C.CHASSIS.CH_NUM, C.ENGINE.CC FROM CAR C;
```

Object Identifiers in Oracle

- Each object instance is given an OID by the system.
- These can be accessed using the 'REF' data type
- Rather trivially:

```
select ref(s) from pgrad s where s.pg_id = 's0067453' ;
```

OIDs are System Defined Pointers

- The result of the query
 - `select ref(s) from pgrad s where s.pg_id = 's0067453' ;`
- is a pointer value (try it)
- These values are assigned for each row of a table which is declared as being OF TYPE ..

Associations using OIDs

- More usefully, we can build associations between object types
- e.g. All post graduates have a supervisor

```
CREATE OR REPLACE TYPE PGRAD_T AS OBJECT (  
    PG_ID          VARCHAR2(8),  
    ST_MODE        VARCHAR2(8),  
    SUPER          REF LECTURER_T,  
    STUDENT STUDENT_T);  
/
```

REF data type

- The attribute SUPER can take values drawn from the domain of Object Identifiers (OIDs) for object instances of the type LECTURER_T (i.e. rows of data in a table declared of type LECTURER_T)
- Such instances will be found in the LECTURER table (defined as being of type LECTURER_T)

Implementation

```
CREATE TYPE LECTURER_T AS OBJECT (  
  ST_ID    VARCHAR2(8),  
  PERSON   PERSON_T);  
/  

```

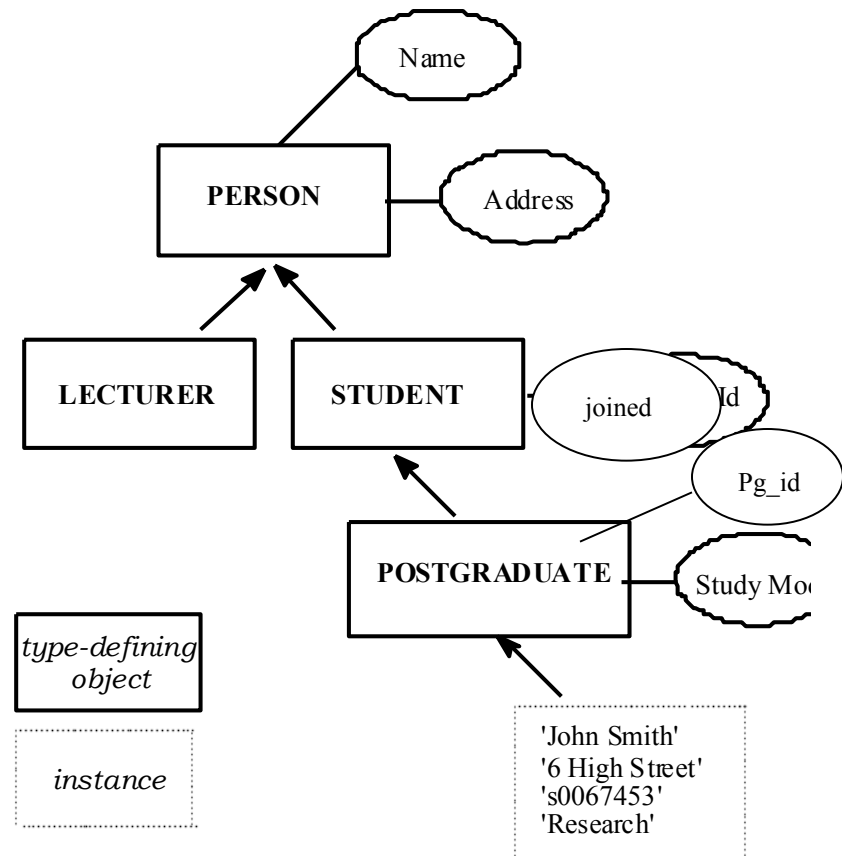
```
CREATE OR REPLACE TYPE PGRAD_T AS OBJECT (  
  PG_ID    VARCHAR2(9),  
  ST_MODE  VARCHAR2(15),  
  SUPER    REF LECTURER_T,  
  STUDENT  STUDENT_T);  
/  

```

```
CREATE TABLE LECTURER OF LECTURER_T(  
  ST_ID    PRIMARY KEY);
```

```
CREATE TABLE PGRAD OF PGRAD_T(  
  PG_ID    PRIMARY KEY);
```

```
INSERT INTO LECTURER VALUES(  
  'P0068892',  
  PERSON_T( 'Bob Champion',  
            '73 Turing Road' ));
```

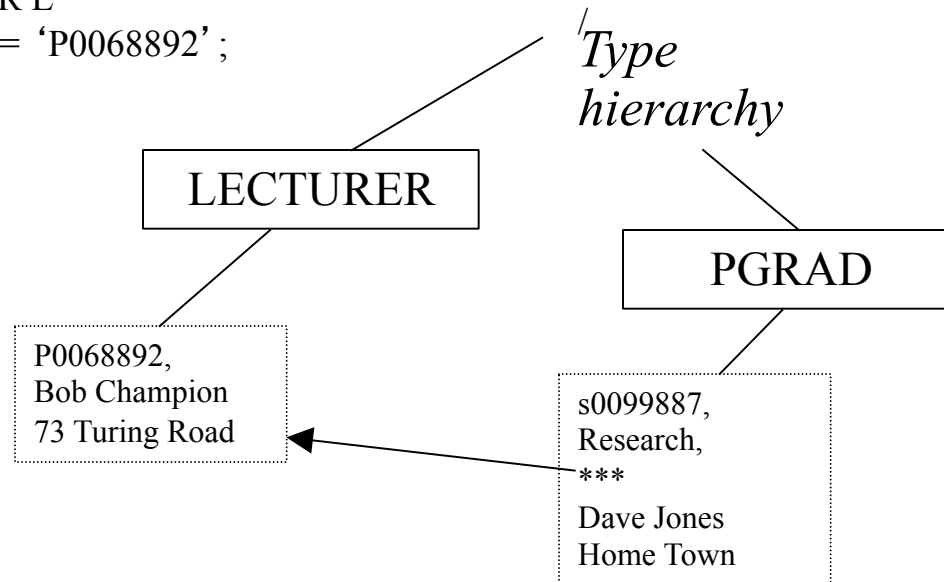


Inserting data to REF data types

```
INSERT INTO PGRAD (pg_id, st_mode, super, student) /
SELECT
    'S0099887',
    'Research',
    REF(L),
    STUDENT_T(' 10-NOV-99',
    PERSON_T( 'Dave Jones', 'Home Town' ))
FROM LECTURER L
WHERE L.ST_ID = 'P0068892';
```

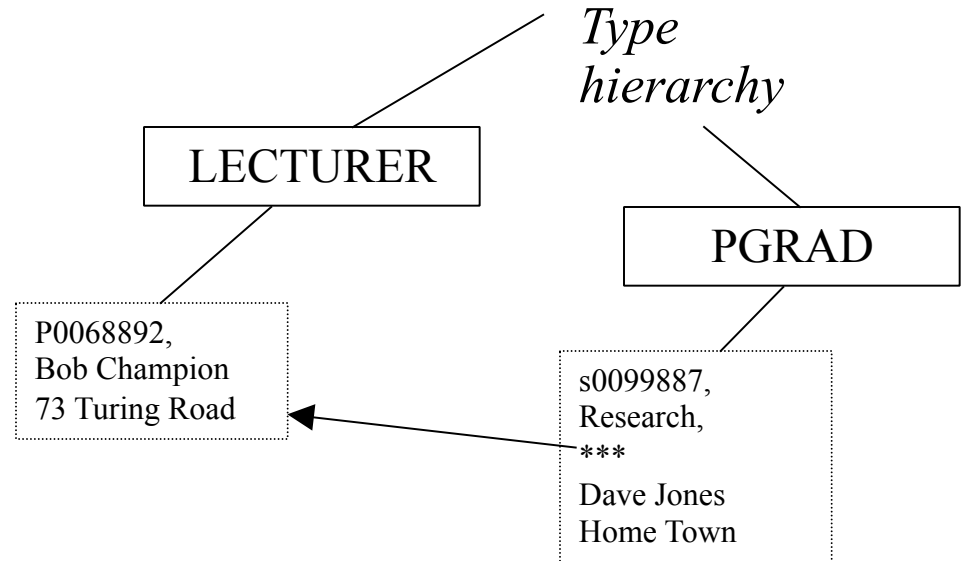
```
CREATE TYPE LECTURER_T AS OBJECT (
    ST_ID    VARCHAR2(8),
    PERSON   PERSON_T);
```

```
CREATE OR REPLACE TYPE PGRAD_T AS OBJECT (
    PG_ID    VARCHAR2(9),
    ST_MODE  VARCHAR2(15),
    SUPER    REF LECTURER_T,
    STUDENT  STUDENT_T);
```



Retrieving Data from REF associations

“List the names and staff numbers of all supervisors of research mode postgraduates.”



```
SELECT L.ST_ID, L.PERSON.NAME
FROM LECTURER L, PGRAD P
WHERE P.ST_MODE = 'Research'
      AND P.SUPER = REF(L);
```

DESIGN

ER Modelling & REFs

- Entity Relationship modelling for relationships can be used with OO databases.
- Many-to-many relationships must still be decomposed.
- One-to-many relationships are implemented using the REF type on the many side (pointing to the one side)

OO Database Design

- An OO database schema is defined in terms of object classes
- These include the data (or attributes) of the class, and its methods
- Two key models of an OO database are the type hierarchy and the object relationship model

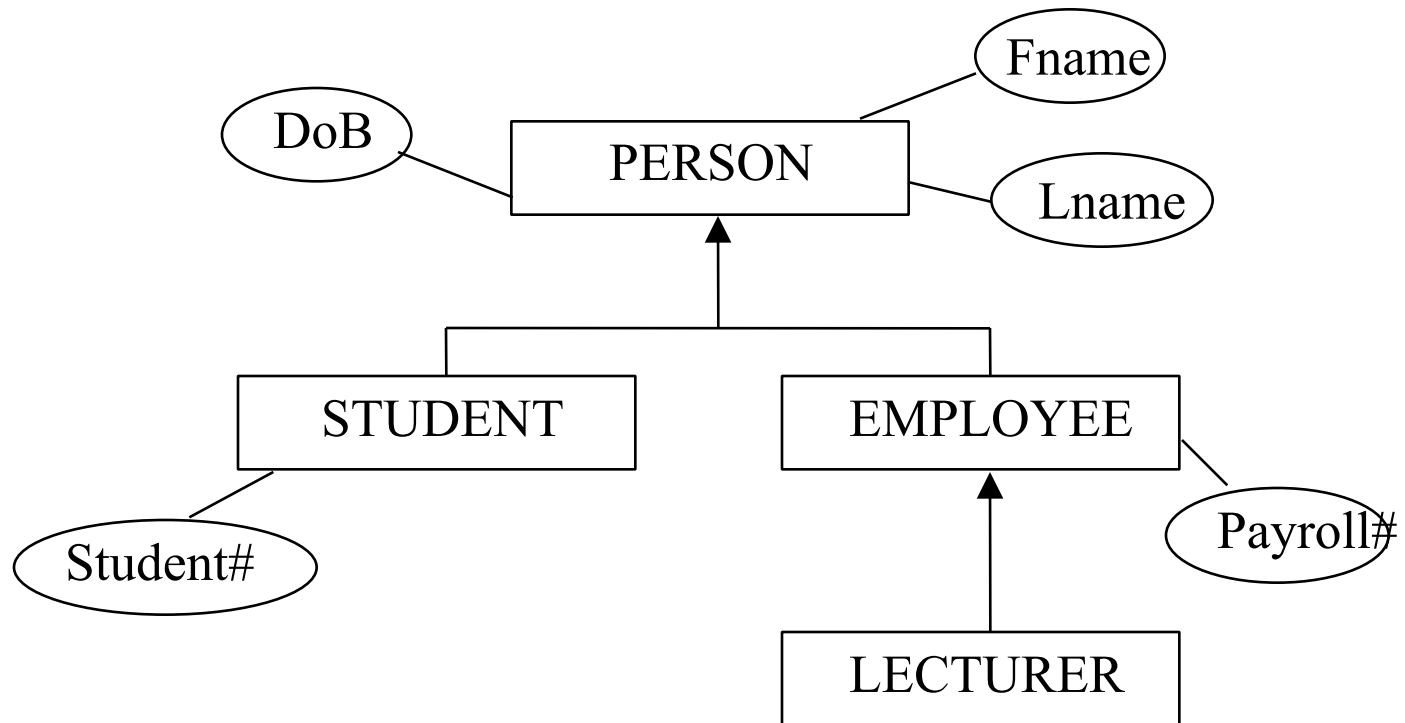
Designing Databases

- Guidelines for deriving object classes
- Relationship definition using 'Pure' OO & Oracle OR
- Populating the database
- Simple data retrieval
- Definition of methods (next week)

Example Domain

- University personnel database
- To contain information about all people employed/taught by the university
- ‘Rules’ for classes same as for entities
- For OO databases, start with any hierarchies of classes

'First attempt' Class Hierarchy



Build Hierarchies First

- Aim is to capture the natural 'is-a' properties of objects
- Higher level types can be reused for many lower levels (e.g. employee)
- Tables are created for leaf nodes only - assuming 'total' partitioning of the subtypes
- Leaf nodes need their own primary keys

Object Classes - OO

- OBJECT subtype PERSON [
 [Fname, STRING],
 [Lname, STRING]
 [Dob, DATE]].
- PERSON subtype EMPLOYEE [
 [Payroll#, STRING]].


OO - Notes on example 1

- Subtypes can be defined as shown
 - all attributes are inherited from the supertype
- Methods may be defined for each type - this is dealt with later
- Note the ‘proper’ subclass definition

Object Classes – Oracle OR

```
CREATE OR REPLACE TYPE NAME_T AS OBJECT(  
    COL VARCHAR2(50))/
```

```
CREATE OR REPLACE TYPE PERSON_T AS OBJECT(  
    LNAME NAME_T,  
    FNAME NAME_T,  
    DOB DATE)/
```



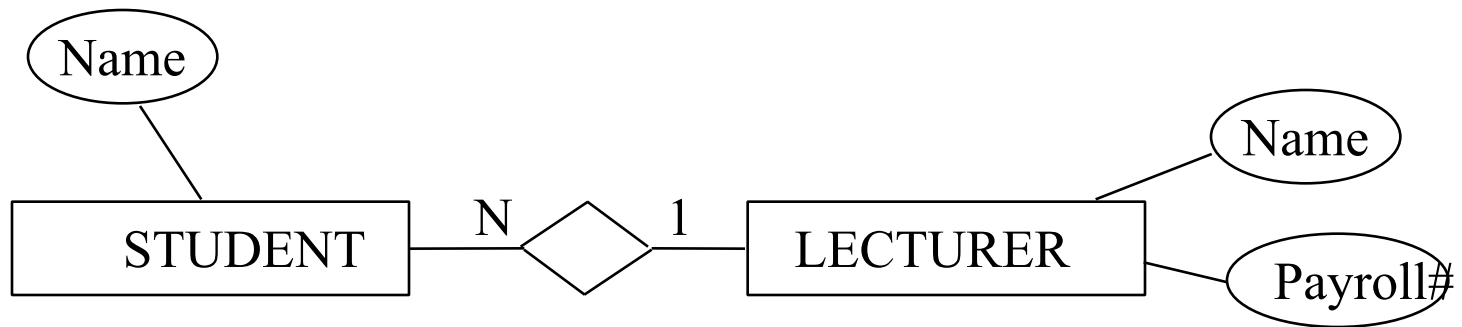
Column type

```
CREATE OR REPLACE TYPE EMP_T AS OBJECT (  
    PAYROLL_ID VARCHAR2(10),  
    PERSON PERSON_T)/
```

Relationships in Pure OO Databases

- Add reference attributes for each binary relationship
- The attributes may be added to both object classes (as inverses of one another)
- Use single valued attributes on the '1' side of the relationship; or set valued for the 'N' side.

Example



‘A student is tutored by one lecturer’

‘An lecturer tutors many students’

Relationships – Pure OO databases

- OBJECT subtype LECTURER [
[EMP#, STRING],
[ENAME, STRING],
[TUTEES, SSET]].
- SET subtype SSET member STUDENTS.
- OBJECT subtype STUDENT [
[NAME, STRING],
[TUTOR, LECTURER]].

Notes on Oracle

- The SET option is not available.
- Note that for 1-to-many relationships, the obvious structure is adequate.
- HOWEVER, there is still a problem with integrity using REF; since ‘dangling’ references are not checked
- I.e. It is possible to delete a LECTURER and leave references from STUDENT.

Relationships in a hierarchy

- Relationships should be specified for the type at which it is appropriate
- Relationships are inherited in the same way as attributes
- e.g. Suppose we wish to specify the all students have a tutor, but only post-graduate students have a supervisor.

OR Relationships – Further Example

```
CREATE TYPE LECTURER_T AS OBJECT (  
  ST_ID VARCHAR2(8),  
  PERSON    PERSON_T);  
/
```

```
CREATE TYPE STUDENT_T AS OBJECT (  
  JOINED    DATE,  
  TUTOR     REF LECTURER_T  
  PERSON    PERSON_T);  
/
```

```
CREATE TYPE PGRAD_T AS OBJECT (  
  PG_ID VARCHAR2(9),  
  ST_MODE VARCHAR2(15),  
  SUPER    REF LECTURER_T,  
  STUDENT  STUDENT_T);  
/
```

```
CREATE TABLE LECTURER OF LECTURER_T(  
  ST_ID    PRIMARY KEY);
```

```
CREATE TABLE PGRAD OF PGRAD_T(  
  PG_ID    PRIMARY KEY);
```

Retrieving Data

Get the Staff number and Name of the tutor of postgraduate student, pg_id = 'pg563542'

```
SELECT L.ST_ID, L.PERSON.NAME  
FROM LECTURER L, PGRAD P  
WHERE P.PG_ID = 'pg563542'  
AND P.STUDENT.TUTOR = REF(L);
```

METHODS

Method Specification

- Include appropriate methods for each class
- These are not shown on the data model
- They must be determined from the processing requirements of the database
- Some examples follow of methods in 'pure' OO

OO - Methods - Creating a set of employees

- ESET has been defined as an object class of type SET;
- The members of the set are OIDs of EMPLOYEES
- We can use an instance of this class to collect the OIDs of all EMPLOYEE objects in the database

OO - Creating a set of employees

`OID_OF_SET_OF_ALL_EMPS := ESET NEW.`

- `OID_OF_SET_OF_ALL_EMPS` is a program variable which is assigned the OID of an empty set of `EMPLOYEE` OIDs
- We can now define a method that applies to objects of class `ESET`, with the effect of adding a new employee to the set

OO Method - Adding a new employee

METHOD : ESET

ADD_EMP# : EMP#_PARAM

ADD_ENAME : ENAME_PARAM

| EMP_OID |

EMP_OID := EMPLOYEE NEW.

EMP_OID PUT_EMP# : EMP#_PARAM,

EMP_OID PUT_ENAME : ENAME_PARAM.

SELF ADD : EMP_OID.

external name

parameters

internal name

local variable

'put' methods

builtin method
'ADD'

OO Method - Joining a union

- We require a method to take an existing EMPLOYEE object, and add them to an existing UNION object
- Assume that:
 - the emp# of the employee is 'e009'
 - the name of the union is ' UGH' and that a program variable `OID_OF_UGH` exists
- Message is - `OID_OF_UGH ADD_EMP : 'e009'`

OO Method Definition

METHOD : UNION

ADD_EMP : EMP_PARAM

|EMP_OID, MEMBERS_OID|

EMP_OID := OID_OF_SET_OF_ALL_EMPS DETECT : [:EX |
EMP_PARAM = EX GET_EMP#].

MEMBERS_OID := SELF GET_MEMBERS

MEMBERS_OID ADD EMP_OID.

OO Notes on the method (a)

METHOD : UNION

ADD_EMP : EMP_PARAM

- Anonymous method applicable to objects of class UNION, with one parameter ADD_EMP (external name)

|EMP_OID, MEMBERS_OID|

- Two local variables

OO Notes on the method (b)

EMP_OID := OID_OF_SET_OF_ALL_EMPS DETECT : [:EX
| EMP_PARAM = EX GET_EMP#].

- DETECT is a builtin method for any SET class
- EX is a range variable over the specified set
- GET_EMP# is a method for the EMPLOYEE class, returning the value of EMP#.
- The result of the DETECT is to return the OID of the first object with an EMP# which matches EMP_PARAM

OO Extra notes

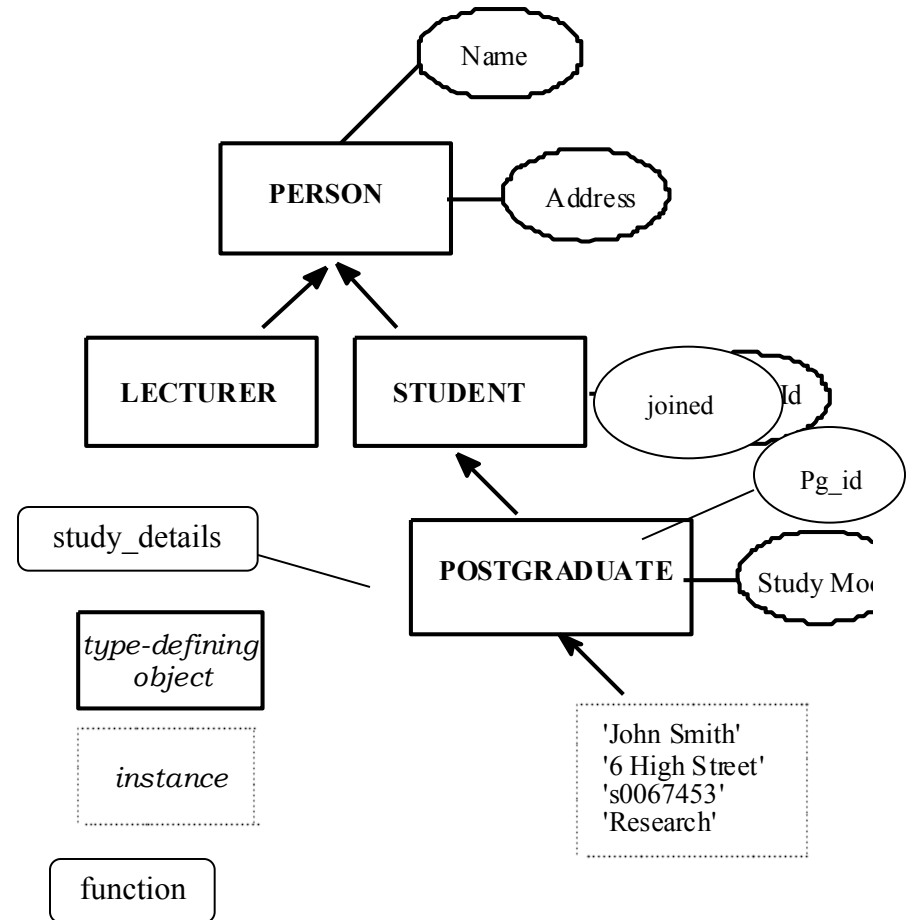
- The last method gives a feel for data retrieval in pure OO
- SELECT can be used to return the OID of the set of all OIDs which a range variable matches
- OO databases are processed using ‘record-at-a-time’ languages
- Loops may be specified

Methods in Oracle

- ‘member functions’ provide facilities for adding methods to object types
- Code for the functions may be added in SQL, plus traditional programming languages
- Function names can be used as extra attributes for the instances

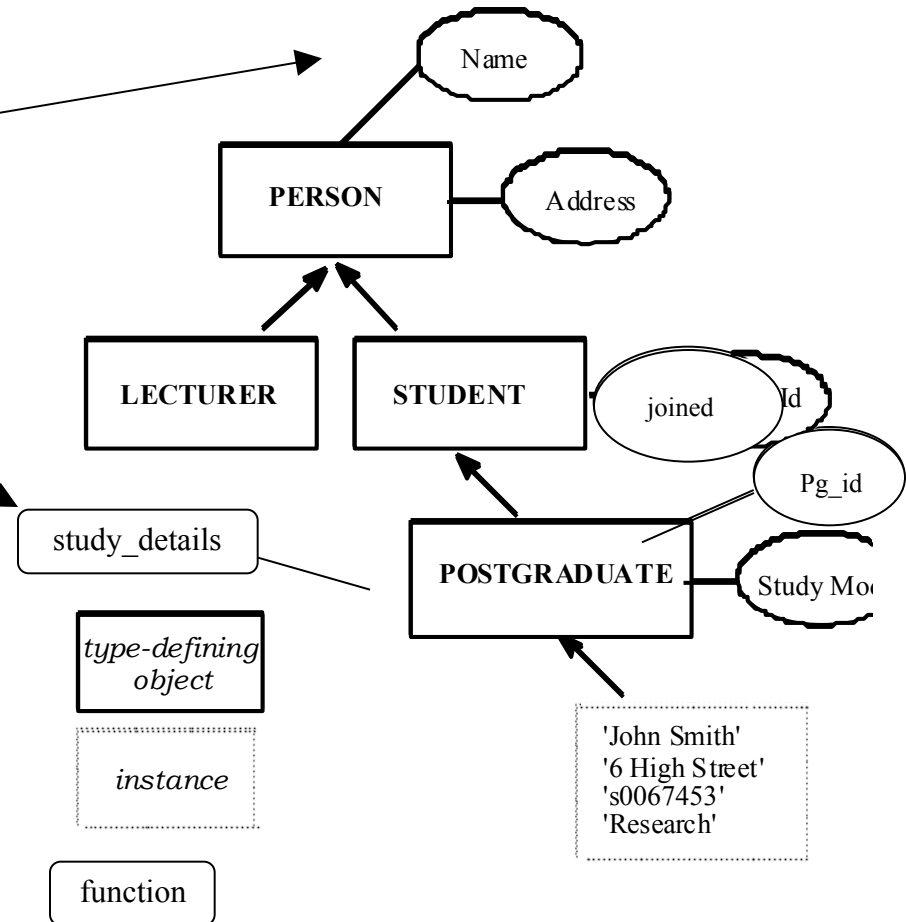
OR Method Specification

```
CREATE OR REPLACE TYPE PGRAD_T AS OBJECT (  
  PG_ID    VARCHAR2(8),  
  ST_MODE  VARCHAR2(8),  
  SUPER    REF LECTURER_T,  
  STUDENT  STUDENT_T  
  MEMBER FUNCTION STUDY_DETAILS  
  RETURN VARCHAR2);  
/  
  
CREATE TYPE BODY PGRAD_T AS  
  MEMBER FUNCTION STUDY_DETAILS RETURN  
  VARCHAR2 IS  
  BEGIN  
    RETURN (SELF.PG_ID || ' ' || 'Study mode: '  
           || SELF.ST_MODE);  
  END STUDY_DETAILS;  
END;
```



OR Method Calls

```
SELECT P.STUDENT.PERSON.NAME,  
P.STUDY_DETAILS()  
FROM PGRAD P;
```

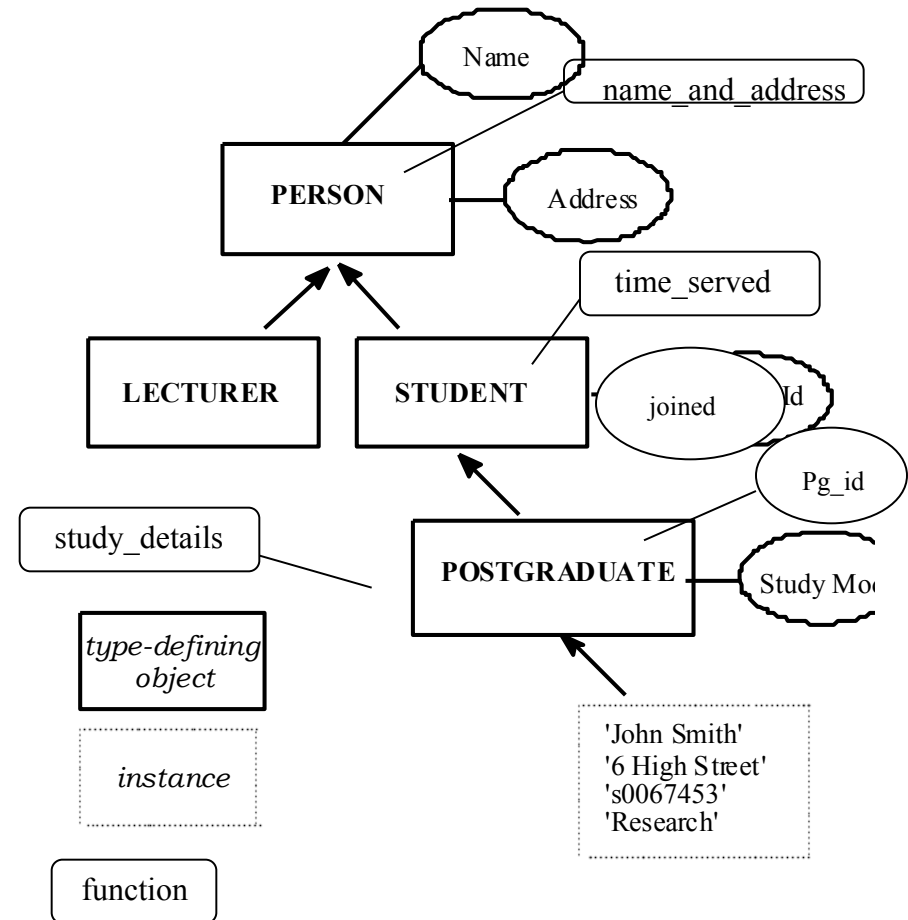


OR Method Inheritance

```
CREATE TYPE PERSON_T AS OBJECT (
  NAME    VARCHAR2(10)
  ADDR    VARCHAR2(50)
  MEMBER FUNCTION name_and_addr
  RETURN VARCHAR2);
```

```
CREATE TYPE STUDENT_T AS OBJECT (
  JOINED   DATE,
  PERSON   PERSON_T
  MEMBER FUNCTION time_served
  RETURN NUMBER);
```

```
CREATE OR REPLACE TYPE PGRAD_T AS OBJECT (
  PG_ID    VARCHAR2(8),
  ST_MODE  VARCHAR2(8),
  SUPER    REF LECTURER_T,
  PERSON   PERSON_T
  MEMBER FUNCTION STUDY_DETAILS
  RETURN VARCHAR2);
```



OR Specification of Methods

```
CREATE TYPE BODY PERSON_T AS
MEMBER FUNCTION NAME_AND_ADDRESS RETURN VARCHAR2 IS
BEGIN
    RETURN (SELF.NAME || ' ' || 'Address: '
            ||SELF.ADDR);
END NAME_AND_ADDRESS;
END;
```

```
CREATE TYPE BODY STUDENT_T AS
MEMBER FUNCTION TIME_SERVED RETURN NUMBER IS
BEGIN
    RETURN (SELF.JOINED - SYSDATE);
END TIME_SERVED;
END;
```

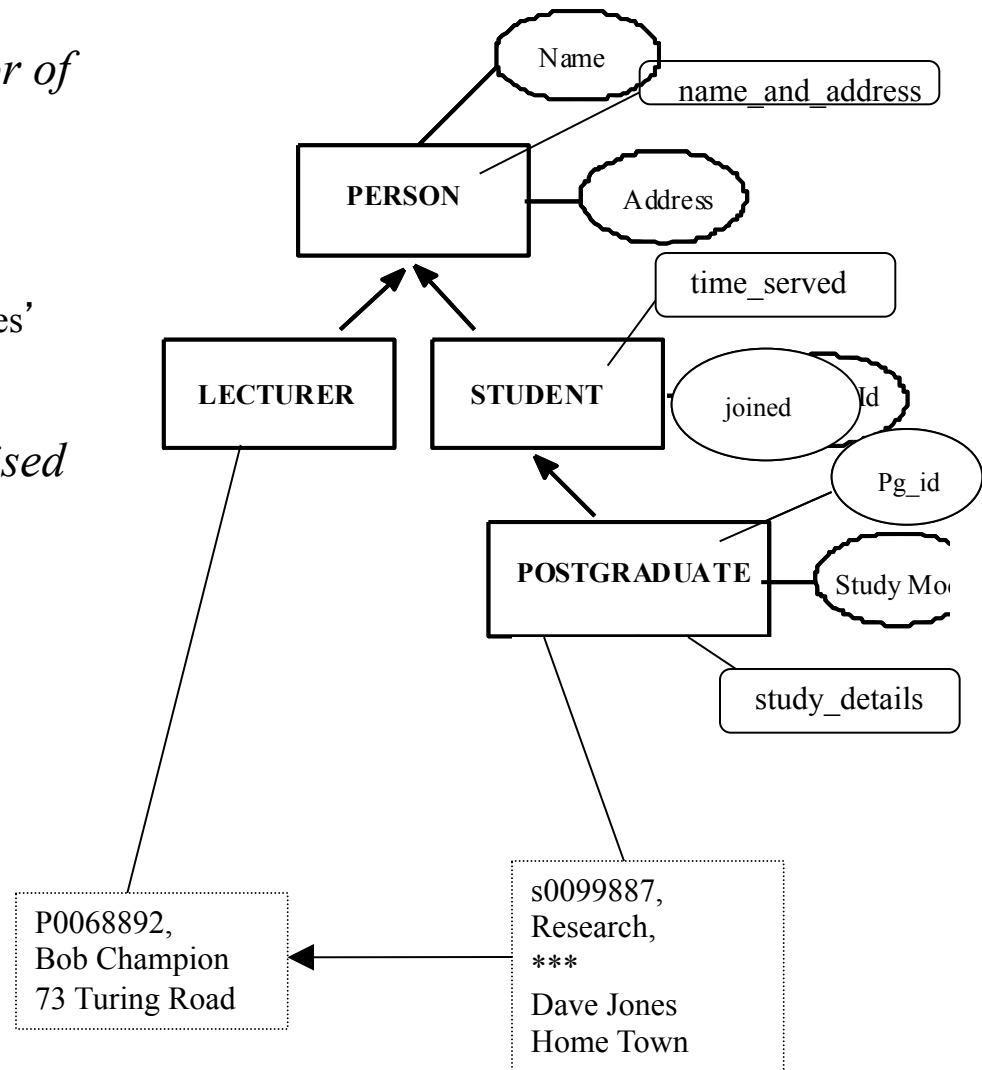
OR Examples on Retrieving Data

Get the name and address of the supervisor of Dave Jones

```
SELECT L.PERSON.NAME_AND_ADDRESS()
FROM PGRAD P, LECTURER L
WHERE P.STUDENT.PERSON.NAME = 'Dave Jones'
AND P.SUPER = REF(L);
```

Get the time served by all students supervised by Bob Champion

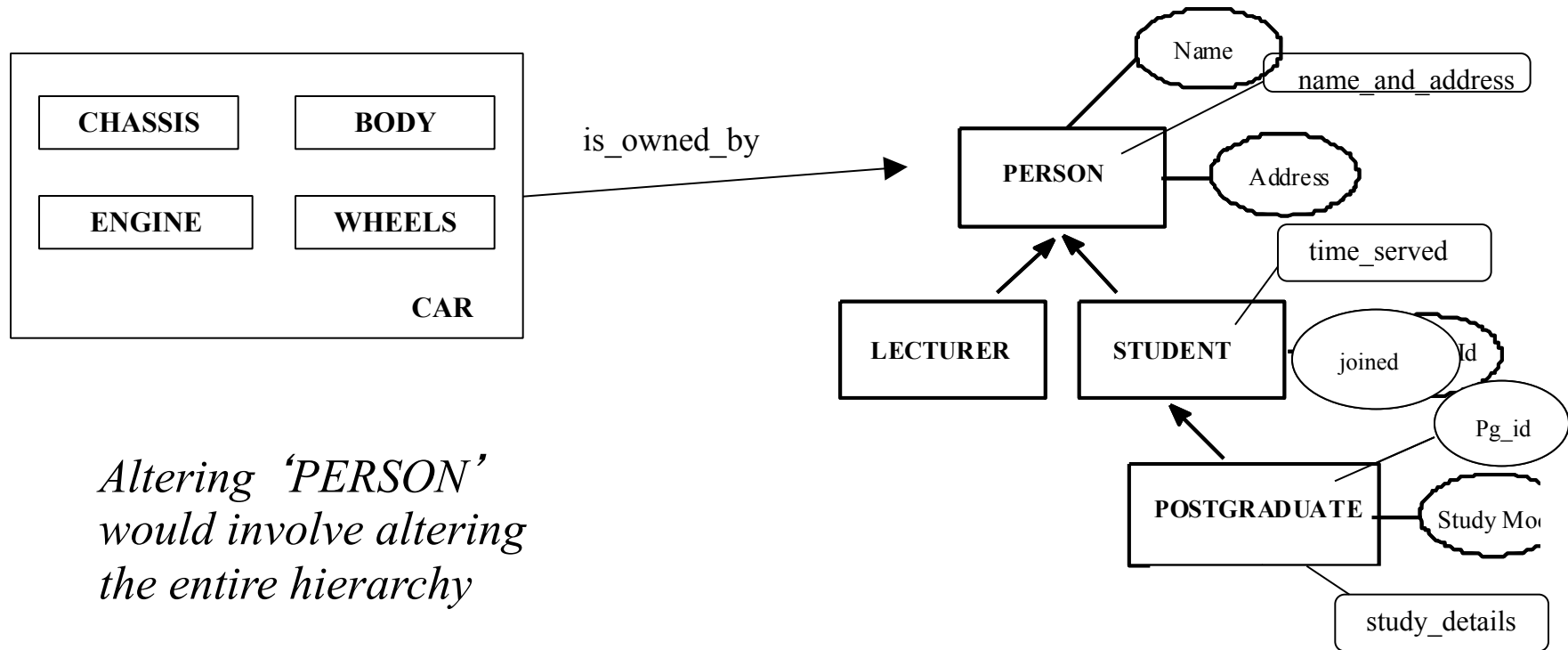
```
SELECT P.PG_ID, P.STUDENT.TIME_SERVED()
FROM PGRAD P, LECTURER L
WHERE L.PERSON.NAME = 'Bob Champion'
AND P.SUPER = REF(L);
```



Example Reuse of Object Structures

- We wish to create a database of cars brought to the university, containing details of the cars and their owners
- A car is owned by one person, but a person may own more than one car
- Note that Oracle8 does NOT allow an object type to be altered if it has dependents

Existing Structures



*Altering 'PERSON'
would involve altering
the entire hierarchy*

*But, fortunately, we
don't have to!*

Re-specification of the CAR Object

```
CREATE TYPE L_CAR_T AS OBJECT(  
    CAR_REG VARCHAR2(10),  
    CHASSIS CHASSIS_T,  
    BODY    BODY_T,  
    ENGINE  ENGINE_T,  
    WHEELS  WHEELS_T,  
    OWNER   REF LECTURER_T);  
/
```

← *Primary key
needed*

```
CREATE TABLE L_CAR OF L_CAR_T(  
    CAR_REG PRIMARY KEY);
```

← *Association with
owner added*

Data Entry

```
INSERT INTO CAR (CAR_REG, CHASSIS, BODY, ENGINE, WHEELS, OWNER)
SELECT
    'Y123 PQR' ,
    CHASSIS_T( 'CTW12543' , 'FORD' ),
    BODY_T( 'RACING GREEN' , 'SALOON' ),
    ENGINE_T(1796),
    WHEELS_T(3, 'SPORTS' )
    REF(L)
FROM    LECTURER L
WHERE   L.PERSON.NAME = 'Bob Champion' ;
```

Important note: the REF data type only stores values of OID of object instances

REF(P) FROM PERSON P cannot be used!

Self-referencing in Oracle

- A method defined on a class, may be executed by any instance of that class
- If the attribute values of the instance are used in the method, we need to use 'self' to refer to the instance (cf. *this* or *self* in OOP)
- Example follows on the next slides:

Example 1 - Methods

```
create type one_t as object(  
  attone varchar2(10),  
  atttwo integer);  
  
/  
  
create type two_t as object(  
  attthree integer,  
  attfour ref one_t,  
  member function prod return integer);  
  
/
```

The Method 'prod'

- The method is required to return the product att3 of the called instance MULTIPLIED BY
att2 of the instance of one_t referenced by the called instance

Example - Local Variables

```
create table one of one_t;  
create table two of two_t;
```

```
create type body two_t as  
member function prod return integer is  
local_one integer;  
begin  
    select o.atttwo into local_one from one o where self.attfour = ref(o);  
    return(self.attthree * local_one);  
end prod;  
end;  
/
```

Some Test Data

```
insert into one values ('test', 10);  
insert into two (attthree, attfour)  
select  
5,  
ref(o)  
from one o  
where o.attone = 'test';
```

Calling the Function

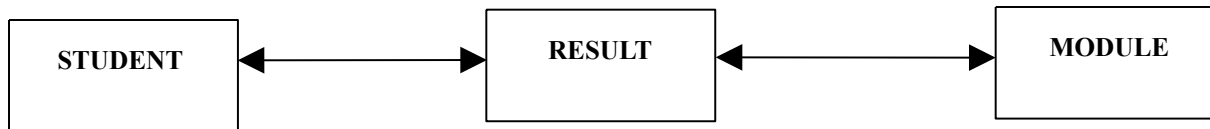
- Select `t.prod()` from two `t`;
- Select `t.prod()` from two `t` where `t.attr3 = 4`;

Example 2

- As part of a university database, we are required to store undergraduate module result.
- Coursework/Exam weights are specified for each module (e.g. 50, 50).
- Students register for a module, and marks are recorded for coursework and exam (each out of 100%)

Design

- The following object types are sufficient for this example:



Types and Tables

```
CREATE OR REPLACE TYPE MODULE_T AS OBJECT (  
  MCODE      VARCHAR2(8),  
  EW        NUMBER,  
  CW        NUMBER);  
/
```

```
CREATE OR REPLACE TYPE STUDENT_T AS OBJECT (  
  UG_ID VARCHAR2(8);  
/
```

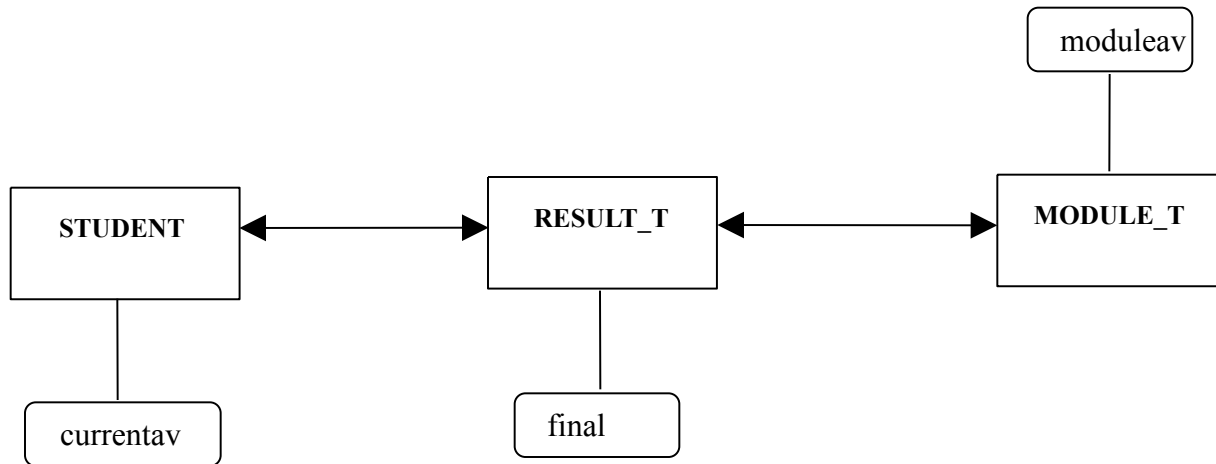
```
CREATE OR REPLACE TYPE RESULT_T AS OBJECT (  
  STUDENT    REF STUDENT_T,  
  MODULE     REF MRUN_T,  
  EM        NUMBER,  
  CM        NUMBER);  
/
```

Adding Functions

- suppose we wish to add:
 - A function to MODULE_T to return the average total mark for all students who took were registered for that module run.
 - A function to STUDENT_T to return the average total mark for all module runs taken by that student.
- We can save time (and effort) by first adding a function to RESULT_T which returns the final mark (based on the weights)

Possible Functions

- Functions added:



- (assume tables student, result, module)

The Function 'final'

```
CREATE OR REPLACE TYPE RESULT_T AS OBJECT (  
    STUDENT REF STUDENT_T,  
    MODULE REF MODULE_T,  
    EM NUMBER,  
    CM NUMBER  
    MEMBER FUNCTION FINAL RETURN INTEGER);  
/  
  
    CREATE TYPE BODY RESULT_T AS  
    MEMBER FUNCTION FINAL RETURN INTEGER IS  
    CW INTERGER;  
    EX INTEGER;  
    BEGIN  
        SELECT M.EW INTO EX  
        FROM MODULE M  
        WHERE SELF.MODULE=REF(M);  
  
        SELECT M.CW INTO CW  
        FROM MODULE M  
        WHERE SELF.MODULE=REF(M);  
  
    RETURN ((SELF.EM * EX / 100) + (SELF.CM * CW * 100));  
    END FINAL;  
END;
```

Using 'FINAL'

- The function 'currentav' can be defined on the type `student_t`
- The function should return:
 `avg(r.final())`
 - where the average is taken over all results obtained by the student for whom the function is called
- `ref(self)` does not work – look for an alternative

Other Functions

- Use this as a starting point
- Add the other functions as specified
- Also add a function to `STUDENT_T` which returns the total number of module runs taken.

Conclusions

- Object Oriented Databases can be used effectively instead of relational databases
- They require a different design approach than relational databases
- They can be equal in performance (given proper design and implementation) but are, inherently, not as efficient
- Using an OO approach is not necessarily worse or better, it depends on many elements.
- Relational databases have been around for a long time and are well understood and implemented. This is not yet the case for OODBs.