Edinburgh Napier UNIVERSITY

Advanced Database Systems SET09107

Object-Relational Databases

Summary

Feedback

Date insertion:

```
Insert Into Table_A Values (
'10-Feb-2016'
);
```

Note: Date ('10-Feb-2016') is not correct

Feedback

Attribute names in referenced types:

```
create or replace type test_ref as object (
employee_r ref employee,
position ref job
);
/
```

Note: Attribute names should be different from referenced type names

Feedback

Use ref():

```
select ref(e)
from employment_ref e
Where e.position.jobtitle = 'manager';
```

Note: the use of dot notation

Feedback

- Use ref() to insert data

  e.g., find the reference for the manager in job_table:

```
insert into employment_table
    select ref(e), ref(j)
    from job_table j, employee_table e
    where e.emp_ID = 2
    and j.job_ID = 1;
```

- The function **ref**() provides the pointers to the objects in the two corresponding tables, which are then inserted into employment_table

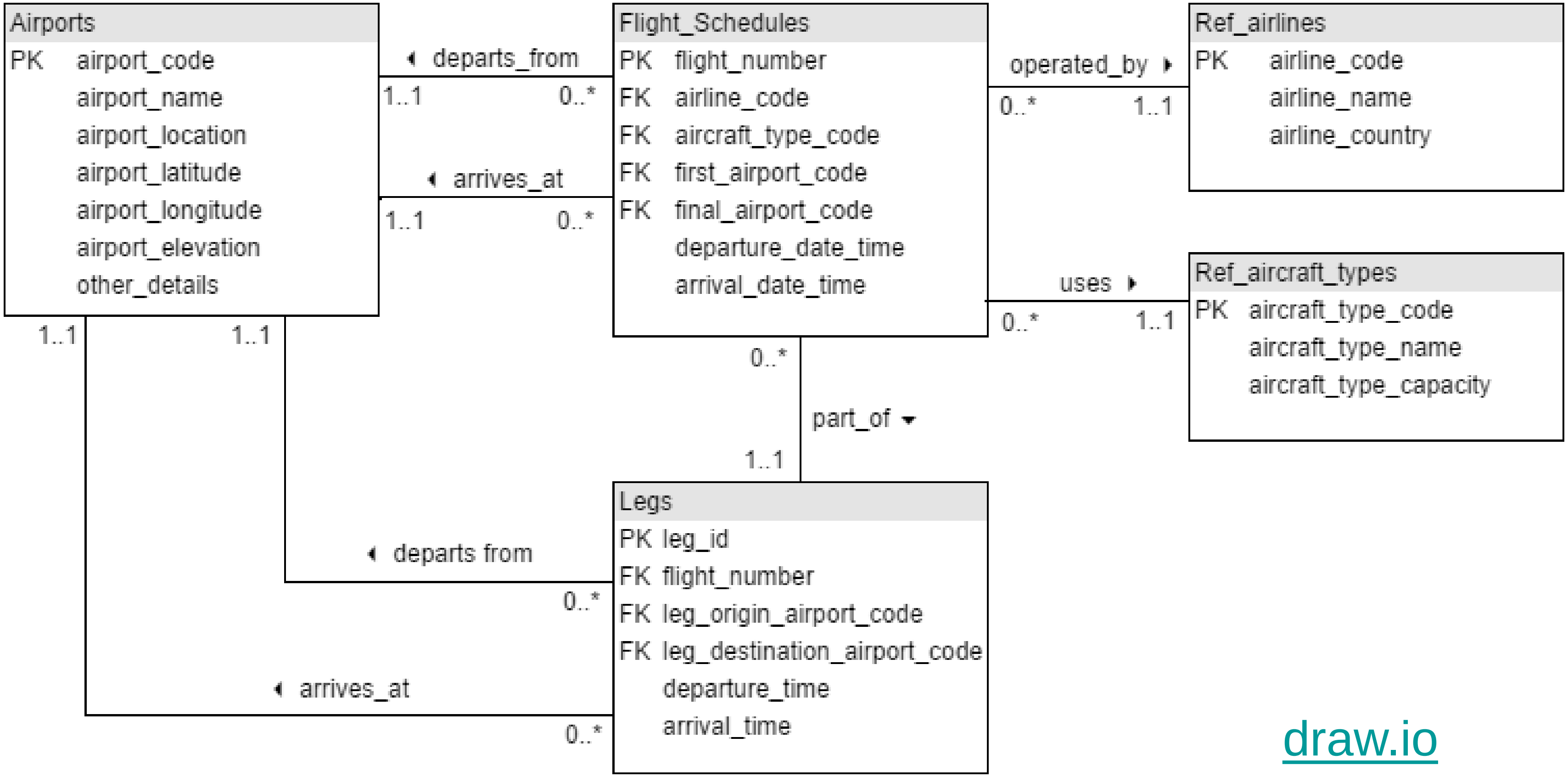
Feedback

Constraints:

```
alter table student_table
    add (constrain surname_const
                        check (name.surname is not null));
```

Note: The “is” is a key word!

The UML equivalent



Airports
PK airport_code
airport_name
airport_location
airport_latitude
airport_longitude
airport_elevation
other_details

Flight_Schedules
PK flight_number
FK airline_code
FK aircraft_type_code
FK first_airport_code
FK final_airport_code
departure_date_time
arrival_date_time

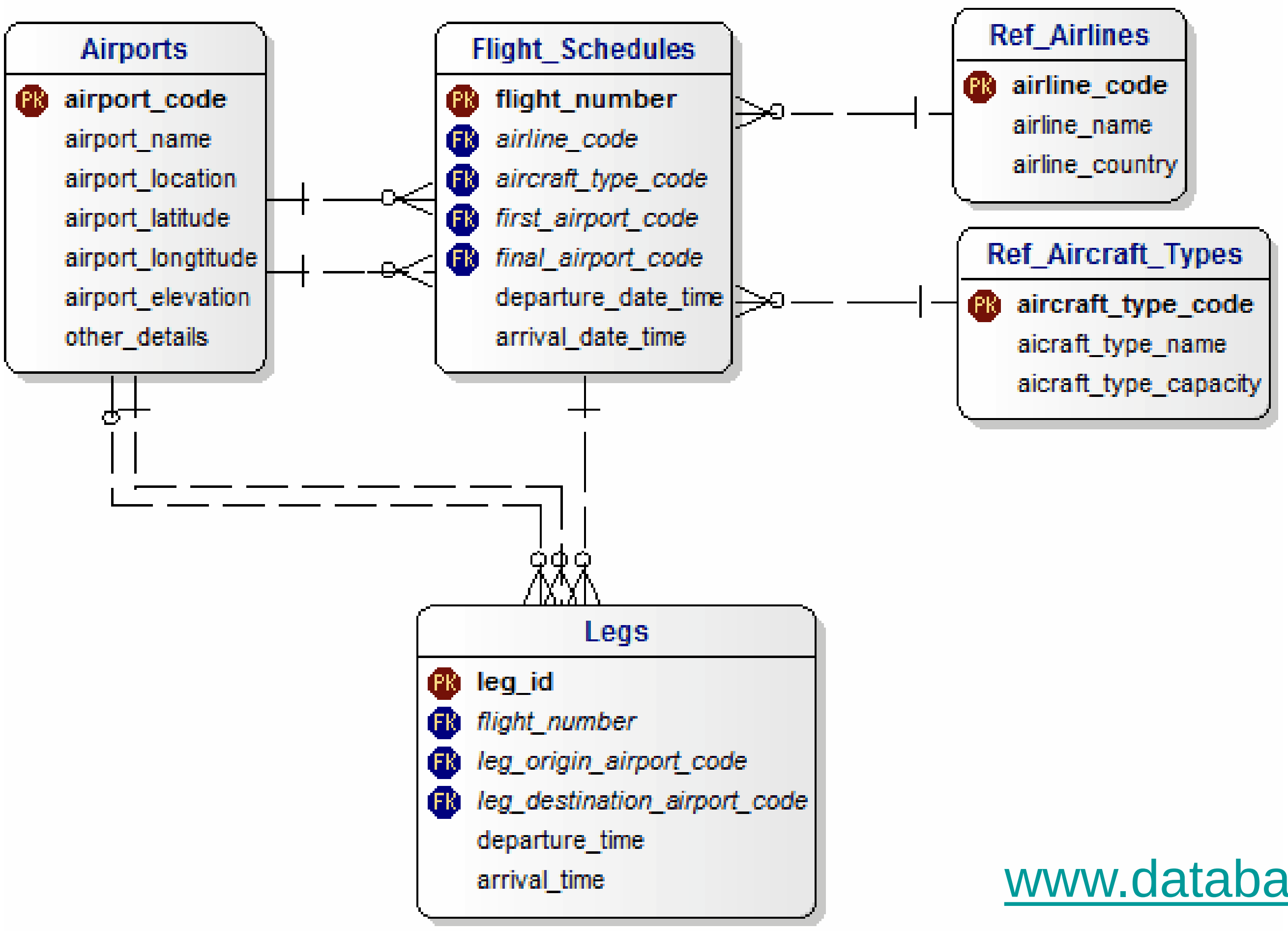
Ref_airlines
PK airline_code
airline_name
airline_country

Ref_aircraft_types
PK aircraft_type_code
aircraft_type_name
aircraft_type_capacity

Legs
PK leg_id
FK flight_number
FK leg_origin_airport_code
FK leg_destination_airport_code
departure_time
arrival_time

draw.io

A small crow's feet model



Airports

- PK **airport_code**
- airport_name
- airport_location
- airport_latitude
- airport_longtitude
- airport_elevation
- other_details

Flight_Schedules

- PK **flight_number**
- FK *airline_code*
- FK *aircraft_type_code*
- FK *first_airport_code*
- FK *final_airport_code*
- departure_date_time
- arrival_date_time

Ref_Airlines

- PK **airline_code**
- airline_name
- airline_country

Ref_Aircraft_Types

- PK **aircraft_type_code**
- aicraft_type_name
- aicraft_type_capacity

Legs

- PK **leg_id**
- FK *flight_number*
- FK *leg_origin_airport_code*
- FK *leg_destination_airport_code*
- departure_time
- arrival_time

www.databaseanswers.org

Contents

- Structured Types
- Subtypes & Inheritance
- References
- Methods
- Constraints
- Collections

Structured Types

- Structured types can be declared:

```
create type Name as object
        ( firstname varchar2(20),
        surname varchar2(20))
        final
create type Address as object
        (street varchar2(20),
        city varchar2(20),
        postal_code varchar2(8))
        not final
```

  - These are called user-defined types
  - The final specification indicates subtypes are not allowed for this type
  - The not final indicates subtypes are allowed

Types & Composite Attributes

- Use structured types to create composite attributes in a relation
- A table can be created

```
create table people
        ( pname Name,
        paddress  Address,
        dateOfBirth date);
```

- components of a composite attribute can be accessed using a “dot” notation, such as pname.firstname

Types & Tables

- Tables also can be defined as

```
create type peopleType as object
        ( pname Name,
        paddress  Address,
        dateOfBirth date)
        not final
create table peopleTable of peopleType;
```

Insert values

```
Insert into peopleTable
  values
    (Name('John', 'Smith'),
     Address('10 Merchiston', 'Edinburgh', 'EH10 5DT'),
    '21-Feb-89'
    );
```

Access Component Attributes

```
select  p.pname.surname, p.paddress.city
from peopleTable  p;
```

Subtypes-- Cont’d

- A supertype can be changed even after some subtypes have been created

```
alter type peopleType
add attribute (gender varchar2(8)) cascade;
```

- The **cascade** option propagates a type change to dependent types and tables

Inheritance

- Subtypes inherit attributes from their supertypes
- Type Student should have *programme, school* in addition of *pname, paddress* and *dateOfBirth*
- Subtypes can redefine methods by using overriding method in place of method in the method declaration

Reference Declaration – Cont'd

- Define a type employment with a field employee and a field position which are references to types employee and job respectively

```
create type employment as object(
    employee_r ref employee,
    position ref job)
```

- A ref is a logical pointer to an instance object (a tuple, any) in the ref type.
- It makes references behave like foreign keys
- The reference points to object types employee and job respectively, not the relevant tables

References -- Functions

Three functions supporting queries involving objects:

- ref() – takes as its argument a table alias associated with a row of an object table and returns the ref to that object
- value() -- takes as its argument a table alias associated with a row of an object table and returns object instances stored in the object table.
- deref() – take the ref to an object as its argument and returns the instance of the object type

Reference – Example – cont’d

```
SQL> SELECT e.employee.pname FROM employment_ref_table e;

EMPLOYEE.PNAME(FIRST, MIDDLE, LAST)
-----------------------------------------------------------
NAME('John', 'R', 'Smith')
NAME('John', 'R', 'Smith')
NAME('Mary', NULL, 'Miller')
NAME('Mary', 'S', 'Miller')
```

Note: You should be able access any tuple in both emplyee_re and job_ref

```
SELECT e.position.jobtitle FROM employment_ref_table e;
SELECT e.position.salary_amount FROM employment_ref_table e;
```

Reference Functions – Cont’d

- Use **value**() to find object instances in a table

  e.g., find object instances for managers in job_table:

  ```
  select value(j)
  from job_table j
  where jobtitle = 'manager';

  VALUE(J)(JOBTITLE, JOB_ID, SALARY_AMOUNT,
     YEARS_OF_EXPERIENCE)
  -------------------------------------------------------------------
  JOB('manager', 3, 52000, 10)
  ```

- Try **select** * **from** job_table **where** jobtitle = 'manager'

Reference Functions – Cont'd

- Use deref() to return the tuple pointed to by a reference

e.g., find the employee in the employee table

```
select DEREF(p.employee)
from employment p
```

```
DEREF(P.EMPLOYEE)(PNAME(FIRST, MIDDLE, LAST),
    PPHONE(HOMEPH, BUSINESSPH, MOBILE
---------------------------------------------------------------------------
EMPLOYEE(NAME('Paul', 'R', 'Miller'), PHONE('123-4587', '838-4536',
    '73556-12')
, ADDRESS('High St', 'Edinburgh', 'EH3 4RF'), 0)

EMPLOYEE(NAME('Sally', NULL, 'Jones'), PHONE('322-5643', NULL,
    '744-6-56'), ADD
RESS('Princess St.', 'Edinburgh', 'EH1 3AB'), 1)
```

Primary Keys -- Oracle

- Object tables can be altered to have primary keys

```
alter table people
add (constraint personID primary key (person_ID));
```

- *personID* is the name of the constraint
- *person_ID* is the name of an actual column in people table.

Methods Oracle

- Member methods -- instance methods
- Static methods – class methods
- An object type with methods must have a separate type body
- **create type** statement specifies
  - The name of the object type
  - Its attributes
  - Methods
  - Other properties
- **create type body** statement contains the code for the methods that implement the type

Methods Oracle – Cont’d

- **create type** statement

```
create type emp as object(
    name varchar2(20),
    salary number,
    member function giveraise ( percent number) return
    number);
```

Methods Oracle – Cont’d

- **create type body** statement

```
create or replace type body emp as
    Member function giveraise (percent number) return number is
    sal number;
    begin
      sal :=(self.salary+(self.salary*percent)/100);
      return sal;
      End giveraise;
    End;
```

- Note: use := in assignments

Methods Oracle – Cont’d

- **Access methods**

```
Select * from emp_table e
where e.giveraise(20)>60000;
```

- It needs to mention which table this method belongs to, e.giveraise

Constraints for Object Tables

- **Primary key constraint** - is used to identify the primary key for a table. It’s similar to the primary key in relational tables, requiring that the primary columns are unique

- **“Check” constraint** - validates incoming columns at row insert time. For example, It can be ensured that the value for city is one of Edinburgh, Glasgow and St. Andrews.

Constraints – Cont'd

- **Not Null constraint** - is used to specify that a column may never contain a NULL value. This is enforced at SQL *insert* and *update* time.

- **Unique constraint -** is used to ensure that all column values within a table never contain a duplicate entry.

Collections

- Oracle supports two collection data types:
  Varrays and nested tables
- Varrays – are variable-length ordered lists (arrays).
- A maximum size of the array must be specified when an attribute of type varray is defined, but not be changed later.

Collections – Cont'd

- Nested tables – are tables within tables
- In contrast to varrays, nested tables are unordered lists
- To define an attribute as a varray or a nested table, a varray or a nested table type definition must first be created.

Varrays

- Assume that there are no more than 10 phone numbers

```
create type phone_array as varray(10) of
   varchar2(12);
create table company1 (
               name varchar2(20),
               phone phone_array
         );
```

Nested Tables

- There are no limit of phone numbers
- A storage table phone_nt_table, which stores the actual values, must be named, although this table can't be used for anything in query.

```
create type phone_nested as table of varchar2(12);
create table company2 (
          name varchar2(20),
          phone phone_nested
        )
    nested table phone store as phone_nt_table;
```

Note: the table name is **phone_nt_table**

Queries for nested tables

- There are two types of queries
  - Queries that retrieve objects in a nested format showing all their types
  - Queries that show the data in an un-nested format, just the data, not the types

Queries for nested tables – Cont’d

- Select only data, using function **table**() to un-nest tables

```
select c.name, t.*
from company2 c, table(c.phone) t;
```

```
NAME                 COLUMN_VALUE
-------------------- ------------
abc                  243-4758
abc                  485-2534
```

Queries for nested tables – Cont’d

- Oracle provides a default column name, **column_value**, for the column of a nested table that doesn’t have a name

select t.**column_value from** company2 c, table(c.phone) t;

```
COLUMN_VALUE
------------
243-4758
485-2534
455-4758
455-2534
```

References

SQL*Plus User's Guide and Reference Release 11.2 pdf

Database Object-Relations Developer's Guide