Database Systems

Department of Software Engineering,

Faculty of Computing,

Sri Lanka Institute of Information Technology.

Course Introduction

- Introduction to Unit
  - Contents
    - Design and development of object-relational databases
    - Design and develop NoSQL and XML database systems for real world applications
    - Describe the principles and techniques of query optimization, estimate the cost of query plans and database tuning.
    - Recommend suitable transaction and concurrency control solutions for data intensive application.

Course Introduction…(contd)

- Explain the concepts underlying in Distributed and Parallel RDBMS architectures and associate protocols for distributed transaction processing.
- Configure Hadoop framework and supporting tools to execute Map Reduce program model for distributed processing.

Course Introduction… (contd.)

- Contact hours
  - 2 hours lecture/week
  - 2 hours practical/week
  - 1 hour tutorial/week

- Recommended References
  - Ramakrishnan, R. and Gehrke, J., Database Management Systems, 3rd ed., McGraw-Hill
  - Elmasri, R. and Navathe, S.B., Fundamentals of Database Systems, 5th ed., Addison-Wesley.
  - Silberschatz A., Korth H.F. and Sudarchan S., Database Systems Concepts, 3rd ed., McGrawHill , 1996
  - Connolly and Begg, Database Systems: A Practical Approach to design, Implementation and management, 3rd ed., Addison-Wesley
  - Shashank Tiwari, Professional NoSQL, John Wiley & Sons, Inc.

Course Introduction… (contd.)

- Grading

Midterm Test	-	20%
Practical Examination	-	20%
Final Examination	-	60%

- To pass this module, students need to obtain a pass mark in both “Continuous Assessments” and “End of the Semester Examination” components which would result in an overall mark that would qualify for a “C” grade or above.

Course Introduction… (contd.)

- All related course materials on CourseWeb
  - http://courseweb.sliit.lk
  - Enrolment key: **SE3060**
- Contact persons:
  - Prasanna S. Haddela (Lecturer-in-charge)
    - Email: prasanna@sliit.lk (preferred for appointments)

Database Systems

Introduction to Object Relational Database Systems

Lecture - 1

Background

- Relational model (70's):
  - Clean and simple representation and access.
    - Tables, primary and foreign keys, SQL.
    - Good foundation of relational structure, algebra, and normal forms.
  - Swept the DB market in the 1980s.
    - Continues to dominate the DB applications even now.
  - Right for business and administrative data.
    - Tables of atomic attributes adequate to represent information.

Background

- Relational model (70's):
  - Not as good for other kinds of data (e.g., multimedia, networks, CAD).
    - Cumbersome to manage such data with Binary Large Objects (BLOBs).
  - RDB has limitations even in its core application areas.
    - Set valued attributes (e.g., academic qualifications, children of employees).
    - Some logical identifiers replaced by artificial keys (e.g., addresses of properties, multiple attribute keys).
    - ISA Hierarchies of entity sets (e.g., student is a person).

Background

- Object-Oriented models (80’s):
  - Proposed as an alternative to relational model.
    - To overcome its limitations.
  - Complex objects were supported.
    - nested relations.
  - Added DBMS functionality to OO programming environments.
    - Object ID, inheritance, methods.
  - ODMG standards: Object data and query languages
    - ODL & OQL.
  - Made limited inroads in the 1990s, then faded away.

Background

- Object-relational DBMS (mid-90's):
  - Extends relational model to support a broader class of applications.
  - Bridge between relational and OO models.
  - SQL:99 and SQL:2003 standards extended SQL to support the OO model.
  - DBMS vendors (e.g., Oracle, IBM, Informix) have added OO functionality.
    - Adherence to the standard varies.

Limitations of relational model

- No support for set valued attributes
    - Example:
        - Person (ID: string, Name: string, PhoneN: string, childID: string)
        - Key: (ID, PhoneN, childID)
        - Not in 3NF (FD: ID -> Name)

ID	Name	PhoneN	ChildID
111	Joe	4576	222
111	Joe	6798	222
222	Bob	5162	333

1NF to 3NF

ID	Name	PhoneN	ChildID
111	Joe	4576	222
111	Joe	6798	222
222	Bob	5162	333

ID	Name
111	Joe
111	Joe
222	Bob

ID	PhoneN
111	4576
111	6798
222	5162

ID	ChildID
111	222
111	222
222	333

Limitations of RDB: Example

- 1NF relation:
  - Person (ID: string, Name: string, PhoneN: string, childID: string)
- 3NF relations:
  - Person(ID, Name)
  - Phone (ID, PhoneN)
  - ChildOf (ID, childID)
- Query: Find the phone numbers of all of Joe’s grandchildren.
- SQL on both schema need multiple joins.

Set valued attributes

ID	Name	PhoneN	ChildID
111	Joe	{4576, 6798}	{222}
222	Bob	{5162}	{333}

- A more appropriate schema:
  - Person (ID: string, Name: string, PhoneN: {string}, child: {string})
  - Query: Find the phone numbers of all of Joe's grandchildren.
  - Ideally, the query could be written as:
    - Select p.child.child.PhoneN
    - From Person p
    - Where p.Name = 'Joe'
  - Note: Oracle implementation is different from this.

SQL extensions for ORDB

- Add OO features to the type system of SQL.
  - columns can be of new user defined types (UDTs)
  - user-defined methods on UDTs
  - reference types and “deref”
  - inheritance
  - old SQL schemas still work! (backwards compatible)

User Defined Types

- A user-defined type, or UDT, is essentially a class definition, with data fields and methods.
- Two uses:

1. As a rowtype, that is, the type of a relation.
2. As the type of an attribute of a relation.

Object types in Oracle

- Uses object types for both columns and rows of relations.
- Example:

```
CREATE TYPE BarType AS OBJECT (
 name CHAR(20),
 addr CHAR(20) )
 /
```

- Note: in Oracle, type definitions must be followed by a slash (/) to store the type.

Object Type

- An object type has 3 components:
  - A name identifies the object type uniquely within the schema.
  - Attributes model the structure and state of the real-world entity.
    - Attributes can be built-in types or object types.
  - Methods, functions or procedures implement operations.
    - (To be covered later.)

Creating Row Objects

- Type declarations do not create tables.
- Object types used in place of attribute lists in CREATE TABLE statements.
  - Example:

```
CREATE TABLE bars OF BarType;
```

- Each row of the table represents an object of given rowtype.

Inserting Values

- As a multi-column table:

```
INSERT INTO bars VALUES ('Sally''s', 'River
   Rd');
```

- As a single column/object value:
  - Each object type (type defined with AS OBJECT) has a type constructor of the same name.
  - Example:

```
INSERT INTO Bars VALUES(
   BarType('Joe''s Bar', 'Maple St.') );
```

Normal select

- Retrieve as a multi-column table:

```
SELECT * FROM bars;
NAME                    ADDR
--------------------------------------
Joe's Bar               Maple St.
Sally's                 River Rd
```

Select as a Single Column

- Select from bars as a single column table.

```
SELECT VALUE(b) FROM bars b;
VALUE(B)(NAME, ADDR)
---------------------------------------------
BARTYPE('Joe''s Bar  ', 'Maple St.   ')
BARTYPE('Sally''s    ', 'River Rd    ')
```

Column Objects

- Objects that occupy table columns in a relational table.
- Example:

```
CREATE TYPE BeerType AS OBJECT (
 name CHAR(20),
 manf CHAR(20) )
 /
CREATE TABLE menu
 (bar bartype,
  beer beertype,
  price real);
```

  - Menu is a table with two attributes of object types.

Alternative: Object table

```
CREATE TYPE MenuType AS OBJECT (
    bar BarType,
    beer BeerType,
    price real)
/
CREATE TABLE Menu2 OF MenuType;
```

- Menu2 is a table of object type.
- Rows of Menu2 are objects (not so in table Menu).
- Methods can be defined on MenuType and invoked on rows of Menu2.

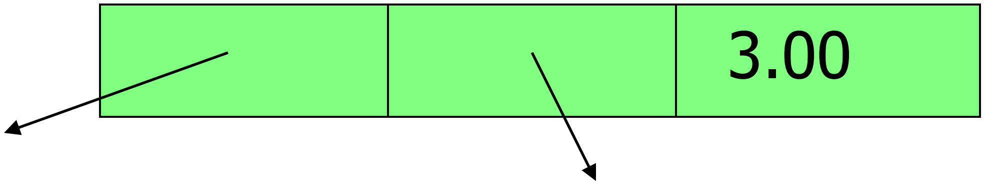
Object References using REF

- If T is a type, then REF T is the type of a reference to T, that is, a pointer to an object of type T.
- Often called an “object ID” in OO systems.
- REF is a built-in data type in Oracle.
- Unlike object ID’s, a REF is visible, although it is usually gibberish.

Example: REF

```
CREATE TYPE MenuType2 AS OBJECT (
    bar         REF BarType,
    beer        REF BeerType,
    price       FLOAT)
/
```

- MenuType2 objects look like:



3.00

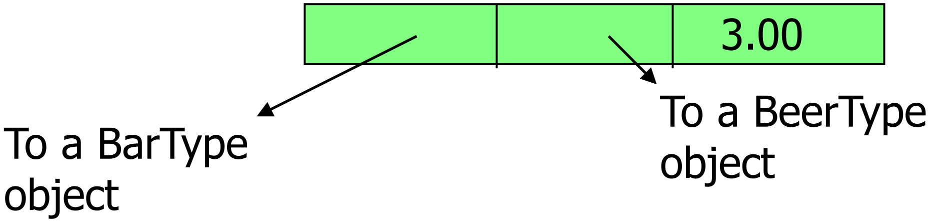
To a BarType object

To a BeerType object

Obtaining REFs

- To get the REF to a row object,
  - select the object from its object table applying the REF operator.
- Example

```
CREATE TABLE Sells OF MenuType2;
```



3.00

To a BarType object

To a BeerType object

```
INSERT INTO sells VALUES(
    (SELECT REF(b) FROM bars b WHERE name='Jim''s'),
    (SELECT REF(e) FROM beers e WHERE name='Swan'),
    2.40);
```

Use aliases to retrieve objects

- To access an attribute of object type, you must use an alias for the relation.
  - Example:

```
SELECT s.Beer.name
    FROM Sells s;
```

- This will not work:

```
SELECT Beer.name
    FROM Sells;
```

- Neither will:

```
SELECT Sells.Beer.name
    FROM Sells;
```

Retrieving with REF Datatype

```
SELECT s.bar.name, s.beer.name, price
   FROM sells s;
```

```
   BAR.NAME     BEER.NAME          PRICE
   ----------------------------------------
   Jim's             Swan               2.40
   Jim's             Bud                3.00
   Sally's           Fosters            2.65
   Sally's           Miller             2.75
```

Dereferencing

- Accessing the object referred to by a REF is called dereferencing the REF.
  - Dereferencing is automatic, using the dot operator.
- Example

```
SELECT s.beer.name
   FROM Sells s
   WHERE s.bar.name = 'Joe''s Bar';
```

Another Example

```
CREATE TYPE person AS OBJECT (
   name VARCHAR2(30),
   manager REF person );
/
CREATE TABLE person_table OF person;
SELECT x.name, x.manager.name
   FROM person_table x;
```

- x.manager.name follows the pointer from the person x to x's manager (who is another person in the table), and retrieves the manager's name.

Scoped REFs

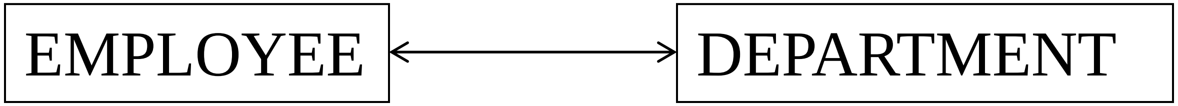
- Example:

```
CREATE type dept_t as object (dno integer, dname
   varchar(12))
/
CREATE TABLE dept_table OF dept_t;
CREATE TABLE dept_loc (
   dept REF dept_t references dept_table,
   loc VARCHAR(20));
```

  - Only objects of dept_table can be values of dept column.

Co-dependent Types

- Types can depend upon each other for their definitions.
  - Example: Object types EMPLOYEE and DEPARTMENT



EMPLOYEE

DEPARTMENT

  - one attribute of EMPLOYEE is the department the employee belongs to and
  - one attribute of DEPARTMENT is the employee who manages the department.

Incomplete Types

```
CREATE TYPE department;
/
CREATE TYPE employee AS
    OBJECT (
    name VARCHAR2(30),
    dept REF department,
    supv REF employee );
/
CREATE TYPE department AS
    OBJECT ( name
    VARCHAR2(30),
    mgr REF employee);
/
```

- CREATE TYPE department; is optional.
- DEPARTMENT is now an *incomplete object type*.
- A REF to an incomplete object type is accepted, so EMPLOYEE type is stored without error.
- Department type is then completed.

Constraints on Object Tables

- Define constraints on an object table just as on other tables
- Example:

```
CREATE TYPE person AS OBJECT (
    pid NUMBER,
    name VARCHAR(25),
    address VARCHAR(50));
/
CREATE TABLE person_table OF person
    ( pid PRIMARY KEY,
     name NOT NULL
   );
```

REF columns

- Oracle does not allow Unique or PRIMARY KEY constraints on REF columns.
- A REF column can be assigned a null value.
- NOT NULL constraint can be specified on such columns.

Null Objects and Attributes

- As in the relational model, a NULL represents an unknown value.
- The following can be NULL:
    - A column value in a table
    - Object
    - Object attribute value
    - A collection, or collection element
- A NULL can be replaced by an actual value later on.

Summary

- ORDB: introduction.
- Object type definitions.
- Creation of row and column objects.
- REF and DEREF operations.
- Constraints on object tables.