Chapter 1: Overview

- *Overview of SQL3 Object Features*
  * User defined data types
  * Oracle8/SQL Methods, Abstract Data Types
  * References, Collections, Large Object
Course Material

- **Course Notes**

- **Butterfly book**
  * Oracle8 Design Tips, D. Ensor & I. Stevenson,

- **Reference Book**
  * A First Course in Database Systems
    - ISBN 0-13-861337-0
Towards object orientation

- Evolutionary path toward object orientation
  - Some projects will wait due to current restrictions

- User-defined types (UDTs)
  - in PL/SQL
  - in tables, row definition on a type

- Object structures within table definitions
  - using VARRAYs, nested tables, and LOBs

- Member functions calls from SQL
  - Similar to column references

- Methods are part of the definition of a type
  - written in PL/SQL
Chapter 2: Object Oriented Features

- *Oracle8 Features*
  - Types
  - Constructing Objects
  - Methods
  - Queries Involving Types
  - Declaring Types For Relations
  - References
  - Converting Relations to Object-Relations

- *Comparison with General Object Model*
Defining Types

- Syntax similar to SQL3
  CREATE TYPE t AS OBJECT (
  list of attributes and methods
  );

  
  * Note slash to process the type definition.

- Example

  * a point type consisting of two numbers:
    CREATE TYPE PointType AS OBJECT ( 
    x NUMBER,
    y NUMBER 
    );

  

Using object data-type

- *Like built-in type in*
  * later declarations of object-types or table-types.

- *Example: we might define a line type by:*
  CREATE TYPE LineType AS OBJECT (
    end1 PointType,
    end2 PointType
  );

- *Example: Create a relation*
  CREATE TABLE Lines ( 
    lineID INT, 
    line LineType 
  );
Dropping Types

- To get rid of a type such as LineType, we say:
  DROP TYPE Linetype;

- However, before dropping a type,
  * we must first drop all tables and other types
    - that use this type.
  * Thus, the above would fail because
  * table Lines still exists and uses LineType.
Constructing Object Values

- **built-in constructors for values of a declared type,**
  - Like C++,
  - constructors bear the name of the type.
- **A value of type PointType is formed by**
  - the word PointType and
  - a parenthesized list of appropriate values.
- **Example: Insert into Lines a with line with ID 27**
  - that ran from the origin to the point (3,4):
    ```sql
    INSERT INTO Lines
    VALUES(27, LineType(
        PointType(0.0, 0.0),
        PointType(3.0, 4.0)
    )
    );
    ```
- **Explanation: Construct a tuple for Lines.**
  - Construct two values of type PointType
  - These values are used the integer 27
Member function Example

• Q? Compare methods with stored procedures.

• Methods vs. stored procedures.
  * Invoke a method any Object of the type
  * Methods may access attributes and methods of its type

• Example
CREATE OR REPLACE TYPE circle AS OBJECT (  
x_pos NUMBER,
y_pos NUMBER,
radius NUMBER,
MEMBER FUNCTION area RETURN NUMBER  
);
/
CREATE OR REPLACE TYPE BODY circle
MEMBER FUNCTION area RETURN NUMBER IS
BEGIN
return(3.1417*radius*radius);
   END;
END;
/

Calling methods in PL/SQL

DECLARE
  my_circle CIRCLE;
  area    NUMBER;
  radius  NUMBER;
BEGIN
  my_circle := circle(2,3,4);
  area    := my_circle.area; -- references a member function
  radius  := my_circle.radius; -- references a stored number
  DBMS_OUTPUT.PUT_LINE ( 'A circle of radius ' || to_char(radius) || ' has an area of ' || to_char(area));
END;
/

Declaring and Defining Methods

- *Type declaration can include methods*
  - defined on values of that type.

- *The method is declared by*
  - MEMBER FUNCTION or MEMBER PROCEDURE
  - in the CREATE TYPE statement
  - Definition code in
  - a separate CREATE TYPE BODY statement.
  - Examples next slide

- *A special tuple variable SELF*
  - refers to the “current” tuple.
  - context should ensure that
  - a particular tuple is referred to.
  - Examples later
Method Declaration - Example

- *Add a length function to LineType.*
  * Apply to the “current” line object,
  * but when it produces the
  * length multiplied by a “scale factor.”

- *Revise the declaration of LineType to be:*
  CREATE TYPE LineType AS OBJECT ( 
  end1 PointType, 
  end2 PointType, 
  MEMBER FUNCTION length(scale IN NUMBER) RETURN NUMBER, 
  PRAGMA RESTRICT_REFERENCES(length, WNDS) 
  );

  /

- *Explanation*
  * mode of arguments - IN, OUT, or INOUT (like ODL).
  * A method can have no arguments, e.g. foo().
  * pragma : length method will not modify database
    - necessary to use length() in queries.
    - WNDS = write no database state
Methods Definition - Example

- All methods for a type are then defined
  * in a single CREATE BODY statement,
  * Note: mode of the argument is not given here.

- Example:
  CREATE TYPE BODY LineType AS
  MEMBER FUNCTION length(scale NUMBER) RETURN NUMBER IS
    BEGIN
      RETURN scale *
      SQRT(((SELF.end1.x-SELF.end2.x)*(SELF.end1.x-SELF.end2.x) +
      (SELF.end1.y-SELF.end2.y)*(SELF.end1.y-SELF.end2.y)));
    END;
  END;
  /
  /
Queries on Defined Types

- **Dot Notation**
  - Values of components of an object, e.g. end1.x
  - More Examples: Find on previous Slide
  - method length() can be called
    - via dot notation on line (type LineType),

- **Queries**
  - Find lengths of lines in Lines
    SELECT lineID, ll.line.length(1.0)
    FROM Lines ll;

- **Restrictions**
  - Dot notation requires alias of a relation name.
  - lineID is a top-level attribute of Lines,
  - length() function is component of line attribute
  - Dropping the ll. or replacing it by Lines. doesn’t work.
Queries on Defined Types

- *Q? What does the following queries do?*
  
  ```sql
  SELECT ll.line.end1.x, ll.line.end1.y
  FROM Lines ll;
  ```

- *Q? What does the following queries print?*
  
  ```sql
  SELECT ll.line.end2
  FROM Lines ll;
  ```

- *Second end of each line printed as PointType(3,4).*
  
  * type constructors used for output as well as for input.
Types Can Also Be Relation Schemas

- **Oracle 8 type can used in two ways**
  - * Columntype, an ADT (as we have done)
  - * SQL3 rowtype

- **Replace the parenthesized list of schema elements**
  - * Ex. Create a table with tuples = a pair of points

```
CREATE TABLE Lines1 OF LineType;
```

- **Same as**
  - end1 PointType,
  - end2 PointType

```
);
```

- **Except that method length is available**
  - * For example, the average length of a line :

```
SELECT AVG(ll.length(1.0))
FROM Lines1 ll;
```
References as a Type in SQL

- *Motivation: Support OID/pointers for efficiency*
  
  * Example
  
  CREATE TABLE Lines2 (  
    end1 REF PointType,  
    end2 REF PointType  
  );

- *REF can create references from actual values.*
  
  * For example,
  
  CREATE TABLE Points OF PointType;

  INSERT INTO Lines2  
    SELECT REF(pp), REF(qq)  
    FROM Points pp, Points qq  
    WHERE pp.x < qq.x;

- *Comparison with Foreign Keys*
References as a Type in SQL

- **Dereferencing: dot notation**
  
  * as if the attribute of reference type
  - were really the same as the value referred to.
  
  * For instance,
    
    ```sql
    SELECT ll.end1.x, ll.end2.x
    FROM Lines2 ll;
    ```

- **Restrictions**
  
  * points referred to must be
  - tuples of a relation of type PointType, e.g. Points
  - No ref to objects appearing in a column of another relation.
  
  * can not invent an object outside of any relation
  
  * and try to make a reference to it.
  
  - Can’t insert `VALUES(REF(PointType(1,2)), REF(PointType(3,4)))`
Nested Tables

• *type of a column can be a table-type*
  
  * A powerful use of object types
  
  * value of an attribute in a tuple can be
    - an entire relation

• *A simple example:*
  
  * Nested Table of built-in type (strings)
    CREATE TYPE items_carried AS TABLE OF VARCHAR2 (100);

    CREATE TABLE store_table (  
      store_id VARCHAR2 (10),
      store_address VARCHAR2 (200),
      store_products items_carried
    );
Nested Tables

- Ex.: Employee(Name, Address)
  * Employee.name is a string
  * Employee.address is a table with 4 columns
  * street, city, state, zipcode.

Name, Address
----, --------

John Doe, ((200 Union St, Minneapolis, MN 55455),
(263 Cleveland Avenue, Roseville, MN 55113))

Jane Doe, ((1200 Silver Lake Road, New Brighton, MN 55112),
(263 Snelling Avenue, Roseville, MN 55113),
(19 West Street, New York, NY 11009))

- - - - -
- - - -
- - - -
Nested Tables - Declaration

• Step 1: Define a type using the AS TABLE OF clause
  
  * Ex. type PolygonType is a relation
  
  * whose tuples are of type PointType;
  
  * which have two numeric components, x and y
  
  CREATE TYPE PolygonType AS TABLE OF PointType;

• Step 2: Define a relation using a PolygonType columns
  
  * Ex. Table Polygons with columns name, and points.

  CREATE TABLE Polygons ( 
      name VARCHAR2(20),
      points PolygonType) 
  
  NESTED TABLE points STORE AS PointsTable;

• Q. What does the last clause say?
Nested Tables - Storage

- **Nested Table "points"**

- **Storage of the ‘‘tiny’’ points table for a polygon**
  * Choice 1: stored as Polygons.points column in each row
    - number tables = 1 (Polygons) + number of Polygons
  * Choice 2: stored in a single table
    - number of tables = 1 (Polygons) + 1 (points)
    - What should be the name of common table?
    - declared via NESTED TABLE clause (PointsTable)

- **Caution about syntax - punctuation**
  * One semicolon ending the CREATE TABLE statement
  * after both
    - the parenthesized list of attributes
    - and the NESTED TABLE clause.

- **Manipulating storage table:**
  * NO SELECTS, eg. (SELECT * from employee_table)
  * ALTER TABLE allowed, eg modify column storage characteristics
  * Can also create an index on a nested table.
Nested Tables - Inserts

- **Inserting tuples in Polygons**
  * type constructor for nested-relation type PolygonType)
  * value of the nested relation =
    - a list of values of the appropriate type

- **Ex. insert a polygon named “square” that consists of**
  * four points, the corners of the unit square.
    
    ```sql
    INSERT INTO Polygons VALUES('square', PolygonType(PointType(0.0, 0.0), PointType(0.0, 1.0),
                                                    PointType(1.0, 0.0), PointType(1.0, 1.0))
    );
    ```

- **Ex. Insert following polygons right angle triangles**
  * with vertices: (0,1), (1,1), (1,0).
  * with vertices: (0,0), (1,1), (1,0).
  * with vertices: (0,0), (0,1), (1,0).
  * with vertices: (0,0), (0,1), (1,1).
Nested Tables - Queries

- **Querying top table**
  
  * Extract rows using other attributes
  
  * Same as before - no change

- **Find all attributes of Polygon named square.**

  ```sql
  SELECT *
  FROM Polygons
  WHERE name = 'square';
  ```

- **Find points in Polygon named square.**

  ```sql
  SELECT points
  FROM Polygons
  WHERE name = 'square';
  ```

- **Q? What will the output look like?**
Nested Tables - Queries

• **Querying nested table - keyword THE**
  
  * get a particular nested relation into the FROM clause
  * "THE" applied to a subquery extracting nested table
  * Q? Which of the above Queries can be used with "THE"?

• **Ex. Find points of the polygon named square**
  - that are on the main diagonal (i.e., x=y).

  ```sql
  SELECT ss.x
  FROM THE(SELECT points
              FROM Polygons
              WHERE name = 'square'
            ) ss
  WHERE ss.x = ss.y;
  ```

• **The nested relation is given an alias ss**
  
  * which is used in the SELECT and WHERE clauses
  * as if it were any ordinary relation.

• **Q? Write queries on nested tables to**
  
  * Count the number of points in Polygon named square
  * Count the number of points in every Polygon
Nested Tables of References

- **Problem: the nested table’s attribute has no name.**
  - Oracle keyword: COLUMN_VALUE

- **Example** - Modify polygons with a nested table of references
  - Schema Definition:
    ```sql
    CREATE TYPE PolygonRefType AS TABLE OF REF PointType;
    /
    CREATE TABLE PolygonsRef (name VARCHAR2(20), pointsRef PolygonRefType)
    NESTED TABLE pointsRef STORE AS PointsRefTable;
    ```

- **Query the points on the main diagonal from 'square’**
  - Problem: 1st line of previous query (i.e. SELECT ss.x)
  - ss has only one column of type REF PointType
  - What are the names of columns of ss ?
  - Soln: Use keyword COLUMN_VALUE

    ```sql
    SELECT ss.COLUMN_VALUE.x
    FROM THE(SELECT pointsRef
    FROM PolygonsRef
    WHERE name = 'square')
    ```
) ss
WHERE ss.COLUMN_VALUE.x = ss.COLUMN_VALUE.y;
Summary of Nested Tables

- **A nested table**
  * is a variation of an Oracle8 collection
  * is a table that appears as a column in another table
  * no size declaration during creation
  * Oracle8 allows any table operation on nested tables

- **Constructor - VALUES**
  * Takes a list of element types

- **New keywords**
  * THE - to access nested tables
  * COLUMN_VALUE - to access REF to nested tables
Exercise with Nested Tables

- Consider a nested table of employee_type via
  - employee_type, employee_table_type and store_table

  CREATE TYPE employee_type AS OBJECT (  
    emp_id NUMBER,  
    emp_lname VARCHAR2 (40),  
    emp fname VARCHAR2 (40),  
    emp_sal NUMBER (10,2)  
  );  
  CREATE TYPE employee_table_type AS TABLE OF employee_type;

  CREATE TABLE store_table (  
    store_id VARCHAR2 (10),  
    store_address VARCHAR2 (200),  
    store_employees employee_table_type)  
  NESTED TABLE store_employees STORE AS employee_table;

- Exercise:
  * 1. Which column has a nested table value?
  * 2. Which table stores nested table value?
3. How many employees can a store have?
Exercise with Nested Tables

• Consider the following INSERT statements

  * Determine the number of stores after each INSERT

  * Determine the number of employees in each store after each INSERT

    INSERT INTO store_table VALUES
    (’Minneapolis’, ’1357 Ventura Way’,
     employee_table_type (employee_type (100, ’Smith’, ’Steven’, 42000.00),
     employee_type (101, ’Fox’, ’Robert’, 28000.00),
     employee_type (102, ’Doe’, ’John’, 16000.00)));

    INSERT INTO THE (SELECT store_employees FROM store_table s
     WHERE s.store_id = ’Minneapolis’)
    VALUES (103, ’Fleeger’, ’Nancy’, 23400.50);

• Advanced Exercises (for Part 4):

  * 1. Does store_table conform to 1st normal form?

  * 2. Redesign store_table for relational databases.
VARRAYs

- **Recall Arrays:**
  * Supported in C, C++, Java, ...
  * Used by application developers

- **Arrays Properties**
  * Ordered sets of objects (elements)
  * Numeric indexes to point to specific elements.
  * All elements in an array must be of the same type.
  * Array’s size depends on the number of elements it contains.

- **Oracle8 Varrays**
  * Implement arrays
  * Support collection data types
  * Do not allocate storage space
  * Variable in size
    - maximum size is defined during its creation
  * Intended Use = small sets of objects
  * Use nested tables for large collection
VARRAYs

• A varray can be used as
  * a datatype of a relational table,
  * an object-type attribute, .IP * or a PL/SQL variable, parameter or function return type.

• Example: DDL statement to create a varray:
  CREATE TYPE pc_components AS VARRAY (100) OF VARCHAR2 (100);

• Creates the varray pc_components
  * a maximum of 100 elements,
  * element type = VARCHAR2, <= 100 characters
VARRAYs - Element Types

- **Element type name can be:**
  
  * A scalar datatype (NUMBER, VARCHAR2, DATE, etc.)
  
  * An REF
  
  * An object type, including those with varray attributes.

- **The element type name cannot be:**
  
  * An object type with a nested table attribute
  
  * A varray type
  
  * A table type (discussed later).
VARRAYs - Constructors

- Constructors for varrays type
  - An implicitly defined constructor method
  - Given the same name as the varray,
  - Must be explicitly called
    - unlike implicit calls for other object type constructors,

- Example: table definition using the varray pc_components:
  CREATE TABLE personal_computers
  (pc_model_no VARCHAR2 (20),
   pc_internal_parts pc_components);
VARRAYs - Instantiation

- *Ex.: inserts values into the personal_computers table:*
  INSERT INTO personal_computers VALUES
  ('BUSPRO 2000',
   pc_components ('EverClear 15 Inch VGA Display', '101 Enhanced Keyboard',
   '5 Bay Tower Chassis', NULL));

- *Explanation*
  * Implicitly defined constructor: pc_components
  * Creates the values for the pc_components varray.
  * Uninitialized elements are set atomically NULL.
  - 4 elements specified, 16 unspecified
VARRAYs - NULLs

- **NULLs vs. NULLs**
  - Developers specified: 4th element’s
  - Constructor set 5th-20th element’s value to NULL.
  - Q? Should there be a difference?

- **Q? What will a SQL SELECT return for a NULL-valued element?**
  - Developer specified: return the NULL.
  - Constructor specified: return a no data found condition.

- **Exercise: Consider the code on next slide.**
  - 1. Which column has a varray value?
  - 2. How many employees can a store have?
  - 3. Determine the number of stores after INSERT
  - 4. Determine the number of employees in each store
VARRAYs of object types - Example Code

- **DDL and DML for Store_table using varrays.**
  
  ```sql
  CREATE TYPE employee_type AS OBJECT
  (emp_id NUMBER,
   emp_lname VARCHAR2 (40),
   emp_fname VARCHAR2 (40),
   emp_sal NUMBER (10,2));
  
  CREATE TYPE current_employees as VARRAY (400) of employee_type;
  
  CREATE TABLE store_table
  (store_id VARCHAR2 (10),
   store_address VARCHAR2 (200),
   store_employees current_employees);
  
  INSERT INTO store_table VALUES
  ('CHICAGO01', '1357 Windy City Way',
   current_employees (employee_type (100, 'Teacher', 'Steven', 42000.00),
    employee_type (101, 'McGowan', 'Robert', 28000.00),
    employee_type (102, 'Zerbe', 'John', 16000.00)));
  ```
VARRAYs - Manipulating Elements

• An SQL SELECT statement will return
  * the contents of the varray as it would any other column.
  * Currently, in SQL you can manipulate whole varrays, but not
  * their individual elements.

• SQL is used primarily to move entire collections of data
  * into and out of database tables,
  * or between client side applications and stored subprograms.

• Within PL/SQL, varrays provide extended procedural capabilities.
  * To manipulate individual elements, PL/SQL programs can
  * compute subscript values and access them by their index pointers.
  * You can pass entire varrays as parameters to PL/SQL subprograms.
Unstructured Data and LOB’s

- **Large Objects (LOBS)**
  * hold up to 4 GByte of raw, binary data,
  * such as images, sounds, and text.

- **Four new datatypes to hold LOBs:**
  * BLOB: unstructured binary data
  * CLOB: single-byte character data
  * NCLOB: fixed-width character data
  * BFILE: data stored outside of the database

- **LOBS are not physically stored with rows**
  * containing other column data.
  * pointers called LOB locators
  * Dangling pointer possible with BFILE

- **Deleting a row with a LOB locator will**
  * delete the corresponding LOB value
Unstructured Data and LOB’s

- **Example**
  
  ```sql
  CREATE TABLE lob_table
  (lob_indicator NUMBER,
   blob_column BLOB,
   clob_column CLOB,
   nclob_column NCLOB,
   bfile_column BFILE)
  LOB (clob_column) STORE AS
   (TABLESPACE clob_tablespace STORAGE
    (INITIAL 5M NEXT 5M PCTINCREASE 50));
  ```

- **Allows multiple LOBs in a table**
  
  * Each LOB an have different Tablespace and
  - Storage Parameters

- **DBMS_LOB package**
  
  * routines to access and manipulate
  * specific parts of LOBs or entire LOBs.
  * read only capabilities to external LOBs.
Converting Ordinary Relations to Object-Relations

- *INSERT statement with embedded query*
  
  * For example, Old table is
  
  ```sql
  CREATE TABLE LinesFlat(
    id INT,
    x1 NUMBER,
    y1 NUMBER,
    x2 NUMBER,
    y2 NUMBER
  );
  ```

- *copy old data into new Lines*

  ```sql
  INSERT INTO Lines
  SELECT id, LineType(PointType(x1,y1), PointType(x2,y2))
  FROM LinesFlat;
  ```
Chapter 3: Object Storage in Oracle8

- *Oracle stores and manages objects in tables*
  * maps objects to tables.

- *Stored with a single column of a row*
  * Built-in types, e.g. number, string, date, ...
  * REFs
  * (leaf-level) varrays with small size (4Kbytes)
    - larger varrays stored separately as BLOBs

- *Other Features*
  * (leaf-level) Nested table types
    - Stored as separate table
  * system-generated object identifier
    - in a hidden column.
Storage of Column Objects

- *A table with a column of an object type*
  * Oracle invisibly adds more columns to the table*
  * for the object type’s leaf-level attributes.*
  * An additional column stores the NULL information of the object*

- *Querying object type column*
  * Oracle call constructor method*
Exercise

• Consider the following Oracle8/SQL3 statements.

  * Count number of hidden columns in Stores table.

  * Create an equivalent physical table in Oracle7.

    CREATE TYPE addr AS OBJECT (  
      street VARCHAR2(10),
      city VARCHAR2(10),
      zip INTEGER
    );

    CREATE TABLE Stores of StoreType
      name varchar(30),
      address addr,
      bestSeller ref BookType
    );
Storage of REF Objects

- **REFs : Physical View**
  
  * Built-in function REF (the row object)
  
  * constructed REF is made up of
    - the object identifier
    - some metadata of the object table,
    - and, optionally, the ROWID.

- **Usage of REF components:**
  
  * ROWID : a hint for efficient access
  
  * Uses ROWID to choose a row;
    - if row.OID = REF.OID then done
    - else use index on OID to get correct row.

- **Size of REF columns**
  
  * 16 to 46 bytes
  
  * Smaller Size by omitting ROWID
  
  * Smaller Size by scoping
Storage of Nested Tables

- The rows of a nested table
  - are stored in a separate storage table.

- Example:
  create table products of product_type
  nested table category store as nest_categories
  nested table inventory store as nest_inventory
  /

- For each nested table in the table definition,
  - the associated storage table contains
  - the rows of all instances of
  - the given nested table in the rows of the parent table.
Customizing Object Properties - Defaults

* Default values

Q? List departments and managers.

CREATE type person AS OBJECT (  
  id NUMBER  
  name VARCHAR2(30),  
  address VARCHAR2(30) );

CREATE TABLE department (  
  d_no varchar2(5) PRIMARY key,  
  d_name varchar2(20),  
  d_mgr person DEFAULT person(1,'Not assigned',null),
);

insert into department  
values('023', 'Accounting', person(2, 'D. Smith', null);

insert into department values('123', 'MIS');
Customizing Object Properties - Constraints

- **Constraints**
  - primary key constraint, foreign key etc.
  - On the leaf-level scalar attributes of a column object
  - with the exception of REFs that are not scoped

- **Examples**

  CREATE type location (  
  building_no NUMBER,  
  city VARCHAR2(40) ) ;

  CREATE type person (  
  ssno NUMBER,  
  name VARCHAR2(100),  
  address VARCHAR2(100),  
  office location ) ;
Customizing Object Properties - Constraints

- **Primary key constraints on person object**
  
  ```
  CREATE TABLE person_extent OF person ( 
    ssno PRIMARY key ) ;
  ```

- **Constraints on scalar attributes of the location objects**
  
  * the dept_loc column of the table.
  
  ```
  CREATE TABLE department ( 
    deptno CHAR(5) PRIMARY key, 
    dept_name CHAR(20), 
    dept_mgr person, 
    dept_loc location, 
    CONSTRAINT dept_loc_cons1 
      UNIQUE (dept_loc.building_no, dept_loc.city), 
    CONSTRAINT dept_loc_cons2 
      CHECK (dept_loc.city IS NOT NULL) ) ;
  ```
Customizing Object Properties - Index

- **Indexes on leaf-level scalar attributes**
  * on an object table
  * on the storage table for a nested table

- **Example**
  
  * an index on the city attribute of the department address.

  ```sql
  CREATE TABLE department (
    deptno CHAR(5) PRIMARY key,
    dept_name CHAR(20),
    dept_addr address );
  
  CREATE INDEX i_dept_addr1
    ON department (dept_addr.city);
  ```
Customizing Object Properties - Privileges

- **Privileges on User Defined Types**
  - * CREATE - can add new types to local schema.
  - * CREATE ANY - can add new types to any schema.
  - * ALTER ANY - can alter types in any schema.
  - * DROP ANY
  - * EXECUTE ANY - can use / reference named types in any schema.

- **Example**
  -- User1 performs the following DDL in the user1 schema:
  CREATE type type1 AS OBJECT ( attr1 NUMBER ) ;
  CREATE type type2 AS OBJECT ( attr2 NUMBER ) ;
  GRANT EXECUTE ON type1 TO user2 ;
  GRANT EXECUTE ON type2 TO user2 WITH GRANT OPTION ;
Exercise

- *Q? Which statements will fail? Why?*

--User2 tries to perform the following:
CREATE TABLE tab1 OF user1.type1;
CREATE type type3 AS OBJECT ( attr3 user1.type2 ) ;
CREATE TABLE tab2 (col1 user1.type2 ) ;
GRANT EXECUTE ON type3 TO user3 ;
GRANT SELECT on tab2 TO user3 ;
GRANT SELECT ON tab1 TO user3;

--User3 tries to perform the following:
CREATE type type4 AS OBJECT (attr4 user2.type3);
CREATE TABLE tab3 OF type4;
GRANT SELECT ON tab1 TO user3;
SELECT * from user2.tab1;
Customizing Object Properties - Privileges

- **Privileges on Object Access**
  
  * Similar to Table access
  * SELECT - can access an object and its attributes FROM the table.
  * UPDATE - can modify attributes of objects in the table.
  * INSERT - can add new objects to the table.
  * DELETE - can delete objects FROM the table.

```sql
CREATE type emp_type (
    eno NUMBER,
    ename CHAR(31),
    eaddr addr_t );
CREATE TABLE emp OF emp_type;
SELECT eno, ename FROM emp;
--Need SELECT previlege for emp table
SELECT VALUE(e) FROM emp e;
--Also Need EXECUTE privilege for emp_type
```
Feature list for Oracle8 Objects

• **Underlying tables required for O8 object model**
  * the object definition is defined for accessing

• **Collections**
  * arrays (e.g., months of year, or days of week)
  * nested tables (e.g., master-details)

• **In O8 object model,**
  * primary key not needed.
  * record (table/row) = an instance of an object

• **Each record has a unique object id (OID)**
  * OID is unique within an enterprise (also the world).
  * OID can be used to explicitly traverse tables
    - without defining join conditions in WHERE clauses.
  * Redundant to primary key
Feature list for Oracle8 Objects

- **OID is similar to current ROWID**
  - BTW, 7.3 rowid format changed in O8
  - a pl/sql function provided for conversion

- **Methods for operating on objects**
  - (e.g., constructor, member, order, user defined)

- **Object model allows developers**
  - to track data and processes as one object

- **Object views (e.g., PO, invoice, work order)**
  - One developer writes methods (e.g. selects, inserts)
  - Other developers save on write join definitions

- **O8 is an active participant in an ORB architecture**
  - Although needs to catch up with DB2, Informix

- **Cartridges:**
  - Image and Video cartridge
  - Spatial cartridge
1-Slide Summary

- Management View
  - Scalable server
  - More users, More data, ...

- Application Developer
  - A step towards object
  - Oracle 8.1 has Java
  - New libraries