Admin matters

- Documents for assignment essays & presentations (Friday)
- next 2 weeks

<table>
<thead>
<tr>
<th>Week</th>
<th>Date</th>
<th>Day</th>
<th>Topic</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>4 Aug</td>
<td>Tues</td>
<td>L5: Object-Relational DBMS</td>
</tr>
<tr>
<td>5</td>
<td>Aug</td>
<td>Wed</td>
<td><strong>L6: Object-Relational DBMS</strong></td>
</tr>
<tr>
<td>7</td>
<td>Aug</td>
<td>Fri</td>
<td><strong>Practical 2:</strong> Object-Relational DBMS (PostgreSQL)</td>
</tr>
<tr>
<td>4</td>
<td>11 Aug</td>
<td>Tues</td>
<td>L7: Object-Relational DBMS</td>
</tr>
<tr>
<td>12</td>
<td>Aug</td>
<td>Wed</td>
<td>L8: Object-Relational DBMS</td>
</tr>
<tr>
<td>14</td>
<td>Aug</td>
<td>Fri</td>
<td><strong>Practical 3:</strong> Object-Relational DBMS (PostgreSQL)</td>
</tr>
</tbody>
</table>
In this lecture

1. Summary of SQL:2008 (standard of SQL with OO data management features)

PostgreSQL

2. Data types: arrays
3. user defined types & arrays
4. Object-oriented (OO) features
5. User defined functions
6. How ORDBMS overcome the weaknesses of the RDBMS (HOMEWORK)
1. Summary of SQL: 2008 (1)

- Definition of data types:

<table>
<thead>
<tr>
<th>Data type / purpose</th>
<th>Statement example</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Built-in type</strong></td>
<td><strong>ROW type</strong></td>
</tr>
<tr>
<td><strong>User Defined Types (UDTs):</strong></td>
<td></td>
</tr>
<tr>
<td>(1) distinct (atomic) types</td>
<td>CREATE TYPE nameType AS VARCHAR(20)</td>
</tr>
<tr>
<td>(2) structured types</td>
<td>CREATE TYPE PersonType AS ( &lt;attribute list&gt; )</td>
</tr>
<tr>
<td><strong>UDT plus:</strong></td>
<td>CREATE TYPE PersonType AS ...NOT FINAL;</td>
</tr>
<tr>
<td>(3) encapsulation</td>
<td>CREATE CONSTRUCTOR METHOD .. CREATE INSTANCE METHOD ....</td>
</tr>
<tr>
<td><strong>UDT plus:</strong></td>
<td>CREATE TYPE StaffType UNDER PersonType AS ......</td>
</tr>
<tr>
<td>(4) inheritance (super&amp; subtypes)</td>
<td></td>
</tr>
<tr>
<td><strong>Built-in:</strong></td>
<td>ARRAY, LIST, SET, MULTISET</td>
</tr>
<tr>
<td><strong>Built-in:</strong></td>
<td></td>
</tr>
<tr>
<td>Collection types</td>
<td></td>
</tr>
<tr>
<td>Reference types &amp; OIDs</td>
<td>similar to OIDs of OODBMS</td>
</tr>
</tbody>
</table>
1. Summary of SQL: 2008 (2)

- Definition of operations on data:

<table>
<thead>
<tr>
<th>Activity &amp; statement type</th>
<th>Statement type</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition of operations on data: User defined routines (UDRs)</td>
<td>(1) functions</td>
</tr>
<tr>
<td></td>
<td>(2) methods, e.g. SELECT S.firstName, S.lastname FROM Staff S WHERE S.isManager;</td>
</tr>
<tr>
<td></td>
<td>(3) operators UDRs can be coded using SQL or HLL e.g. C, C++</td>
</tr>
<tr>
<td>Definition of operations on data: polymorphism on UDRs</td>
<td>Overloading, overriding, static binding, dynamic binding of methods are all supported.</td>
</tr>
</tbody>
</table>
### 1. Summary of SQL:2008 (3)

- Data definition and data access

<table>
<thead>
<tr>
<th>Activity</th>
<th>Statement example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition of database structures</td>
<td>CREATE TABLE Branch (BranchNo CHAR(4), address ROW(.................));</td>
</tr>
<tr>
<td>(1) e.g. using ROW type</td>
<td>CREATE TABLE Person OF PersonType;</td>
</tr>
<tr>
<td>(2) using UDT</td>
<td>CREATE TABLE Staff OF StaffType UNDER Person;</td>
</tr>
<tr>
<td>(3) using UDT and inheritance</td>
<td>--alternative way of inheritance</td>
</tr>
</tbody>
</table>

Storing data in database

- INSERT INTO <table name> ......

Retrieving data from database

- SELECT statement
### 1. Summary of SQL:2008 (4)

<table>
<thead>
<tr>
<th>Activity</th>
<th>Statement example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Definition of database structures</td>
<td>.CREATE TYPE StaffType ( .......... REFERENCE IS SYSTEM GENERATED...);</td>
</tr>
<tr>
<td>Reference types &amp; OIDs (for modeling relationships)</td>
<td>CREATE TABLE Staff of StaffType ( ............);</td>
</tr>
<tr>
<td></td>
<td>CREATE TABLE PropertyForRent ( staffID REF(StaffType) SCOPE Staff</td>
</tr>
<tr>
<td>Special operations for database update:</td>
<td>e.g.</td>
</tr>
<tr>
<td>Trigger: SQL (compound) statement <strong>executed automatically</strong> by DBMS as <strong>side effect</strong> of a modification to named table</td>
<td><strong>CREATE TRIGGER</strong> InsertPropTable <strong>AFTER INSERT ON</strong> PropertyForRent  .....</td>
</tr>
</tbody>
</table>
# 1. Summary of SQL:2008 (6)

<table>
<thead>
<tr>
<th>Activity</th>
<th>Statement example</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stored procedure</td>
<td></td>
</tr>
<tr>
<td>Special data types</td>
<td>Character Large Object (CLOB)</td>
</tr>
<tr>
<td></td>
<td>Binary Large Object (BLOB)</td>
</tr>
</tbody>
</table>
2. Using PostgreSQL in Windows (a)

- pgAdminIII or from Windows command-line

To access query tool:
Tools | Query Tool
2. PostgreSQL – Data types (b)

1. Numeric Types
2. Monetary Types
3. Character Types
4. Binary Data Types
5. Date/Time Types
6. Boolean Type
7. **Enumerated Types**
8. Network Address Types
9. Bit String Types

10. Geometric Types
   - Points
   - Line Segments,
   - Boxes, Paths
   - Polygons
   - Circles

11. Text Search Types
12. XML Type
13. **Composite Types**
14. **Arrays**
15. Object Identifier Types
16. Pseudo-Types
PostgreSQL – user defined types (3a)

- **Array types** similar to HLL array types e.g.:
  - column of a table can be defined as variable-length array.
- **Array elements** may be of:
  - of any built-in type
  - **user-defined base type**,  
  - enum type ,
  - composite type
- e.g. of declaration

```sql
CREATE TABLE salEmp (  
  name text,  
  payByQuarter integer[],  
  schedule text[][];  
) /* 1D array *//* 2D array */
**Array types** similar to HLL array types e.g.:

```sql
CREATE TABLE salEmp (  
    name text,  
    payByQuarter integer[],  
    schedule text[][]);  
```

```
INSERT INTO salEmp VALUES (  
    'Thando',  
    '{20000, 25000, 25000, 25000}',  
    '{ { "breakfast", "consulting" }, {"lunch", "meeting",} }' );
```

OR:

```
INSERT INTO salEmp VALUES  
    (‘Thando’,  
    ARRAY[ 20000, 25000, 25000, 25000 ],  
    ARRAY[ ['breakfast', 'consulting'], ['lunch','meeting'] ] );
```
PostgreSQL – user defined types (3c)

• **Array types** similar to HLL array types e.g.:

```sql
CREATE TABLE salEmp (  
  name text,  
  payByQuarter integer[],  
  schedule text[][]);  
```

• The **current dimensions of any array value** can be retrieved with the `array_dims` function

• e.g. to determine the **dimensions of the ‘schedule’ column**:

```sql
SELECT array_dims(schedule)  
FROM salEmp WHERE name = 'Thando';  
```
PostgreSQL – user defined types (3d)

- **Array types** similar to HLL array types e.g.:
  
  ```sql
  CREATE TABLE salEmp (  
    name text, payByQuarter integer[], schedule text[][]);
  ```

- **other select examples:**
  
  ```sql
  SELECT name FROM salEmp  
  WHERE payByQuarter[1] <> payByQuarter[2];
  ```

  ```sql
  SELECT name, payByQuarter [3] AS Q3pay FROM salEmp;
  ```

  ```sql
  SELECT name, schedule[1:1][1:2] FROM salEmp;
  ```

  ```sql
  SELECT name, schedule[1:2][2] FROM salEmp  
  WHERE name = 'Thando';
  ```
CREATE TABLE CITIES (  
    name text,  
    population real,  
    area int ); /* in sq km */

CREATE TABLE CAPITALS (  
    state char(2))
INHERITS (CITIES);

=> a row of CAPITALS inherits all columns (name, population, and area) from its parent, CITIES.

State CAPITALs have an extra column, state.

In PostgreSQL, a table can inherit from zero or more other tables.
CREATE TABLE CITIES ( name text, population real, area int );
CREATE TABLE CAPITALS ( state char(2) ) INHERITS (CITIES);

INSERT INTO CITIES VALUES ('Johannesburg', 2.5, 2500);
INSERT INTO CAPITALS VALUES ('Pretoria', 1.5, 900, 'GP');

e.g. list the names of all cities including state capitals
SELECT * FROM CITIES ;

e.g. list all the capitals:
SELECT * FROM CAPITALS;

<table>
<thead>
<tr>
<th>name text</th>
<th>population real</th>
<th>area integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Pretoria</td>
<td>1.5</td>
<td>900</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Four categories user defined functions:
(1) query language functions (SQL functions: written in SQL)
(2) procedural language functions
  (functions written in, e.g. PL/pgSQL or PL/Tcl)
(3) internal functions
(4) C-language functions

Two categories of SQL functions:
(1) Row functions: operate on values of a table row
(2) Table functions: operate on whole table
  • are functions that produce a set of rows, made up of:
    • either base data types (scalar types) or
    • composite data types (table rows).
SQL functions use the following syntax:

```
CREATE FUNCTION functionName(argtype1, argtype2, .. ) RETURNS returntype AS
 $$
 SQL statement
 $$ LANGUAGE SQL;
```

**Example:**

```sql
-- DROP FUNCTION populationDensity(real, integer);  if you need to
CREATE FUNCTION populationDensity(real, int) RETURNS int  AS
 $$
 SELECT CAST ( (($1 * 1000000) / $2 ) AS int )
 AS populationDensity;
 $$ LANGUAGE SQL;
```
CREATE FUNCTION populationDensity(real, int) RETURNS int AS $$
SELECT CAST ((($1 * 1000000) / $2) AS int) AS populationDensity;
$$ LANGUAGE SQL;

SELECT name, population, area, 
    populationDensity( population, area )
FROM CITIES ;
The syntax of a PL/pgSQL function is as follows:

```sql
CREATE FUNCTION functionname(argtype1, argtype2, .. )
    RETURNS returntype AS
$$
DECLARE
    --comment 1: declare vars only if necessary
    var1 vartype1, var2 vartype2,.... ;
BEGIN
    -- comment 2: code to compute return value
    RETURN returnvalue ;
    --comment 3: return value must be of returntype
END;
$$ LANGUAGE plpgsql;
```
CREATE FUNCTION popDensity(real, int) RETURNS int AS $$
DECLARE density int;
BEGIN
    density = CAST ( (($1 * 1000000) / $2 ) AS int );
    RETURN density;
END;
$$ LANGUAGE plpgsql;

SELECT name, population, area, popDensity(population, area) FROM CITIES ;
Homework for the student

- How does an ORDBMS e.g. PostgreSQL overcome the weaknesses of the RDBMS?