Práctica 2: Algebra relacional y consultas SQL

Grado en Ingeniería del Software - Bases de Datos

Curso 2015/16

Objetivos

- Tomar contacto con una primera base de datos relacional.
- Practicar la resolución de consultas con el álgebra relacional con el sistema DES.
- Practicar consultas básicas en SQL con SQLDeveloper o/y SQLPlus.

Introducción

En esta práctica se usará en primer lugar el lenguaje formal del modelo relacional que se ha estudiado en clase: el álgebra relacional (AR). Para ello se usará la herramienta educativa DES (des.sourceforge.net). Esta herramienta es compatible con la sintaxis de otra herramienta educativa (WinRDBI, que se puede consultar en el apéndice de "Understanding Relational Database Query Languages" Suzanne W. Dietrich, Arizona State University, 2001, ISBN 0-13-028652-4, y que se puede encontrar en la biblioteca de la facultad). En segundo lugar se usará el lenguaje de consultas estructurado SQL para resolver las mismas consultas que las resueltas con el álgebra relacional.

La base de datos sobre la que realizaremos las consultas responde al siguiente diagrama.

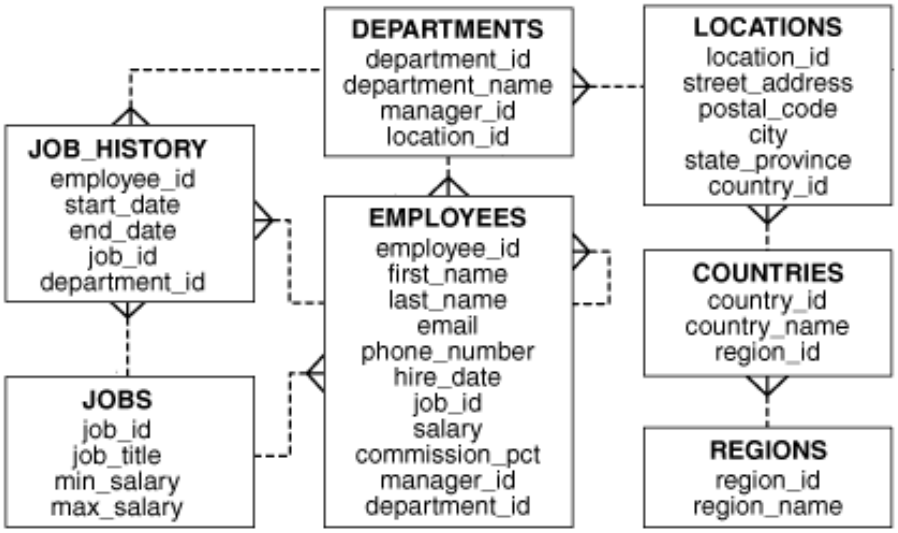

Figura 1. Diagrama UML de la base de datos.

Álgebra relacional con DES

Para poder usar el intérprete del álgebra relacional en primer lugar se debe instalar el sistema DES desde la dirección http://des.sourceforge.net/ según se indican en las instrucciones del sitio web. Al final de este documento se incluye la sintaxis de los distintos operadores (también se puede consultar el apartado dedicado al álgebra relacional en el manual online o en el pdf de DES). A continuación se crearán las tablas para una base de datos con el esquema que aparece en la figura 1.

El siguiente paso consiste en cargar los datos. Para ello se proporciona el archivo **`DES01-crearTablas.sql`** para crear las tablas en DES y el archivo **`DES02-cargarDatos.sql`** para la

inserción de datos en las tablas. Los archivos contienen scripts que pueden ejecutarse a través de la interfaz gráfica (símbolo del "play") o a con el comando **/process** desde la consola. Después se puede emitir una consulta simple para ver que se han cargado los datos. Aquí tienes un ejemplo:

```
DES> /process DES01-creaTablas.sql
Info: Processing file 'DES01-creaTablas.sql' ...
...
DES> /process DES02-datosTablas.sql
Info: Processing file 'DES02-datosTablas.sql' ...
...
DES> select count(*) from employees;
answer($a11:int) ->
{
  answer(107)
}
Info: 1 tuple computed.
```

El sistema DES proporciona la respuesta o resultado de la consulta entre llaves. En el ejemplo anterior, la respuesta es 107, que se corresponde con el número de empleados en la tabla **employees**.

Para resolver los ejercicios conviene probar cada consulta por separado y se deben almacenar todas en un fichero de texto denominado **P2_DES_GISXX.sql** para reproducir posteriormente la sesión.

OJO:

- DES distingue mayúsculas y minúsculas en los identificadores de las relaciones. Usad letras minúsculas para referiros a ellos.
- El modo de entrada predeterminado es por línea (la entrada se procesa en cuanto se pulsa INTRO). Se puede cambiar al modo multilínea con el comando **/multiline on** (la entrada requiere entonces un punto y coma al final, como en SQL).
- El modo de respuesta predeterminado es orientado a conjuntos. Se puede cambiar al modo multiconjunto (como en los sistemas SQL habituales) con el comando **/duplicates on**.

SQL con Oracle

En el apartado 3 de la práctica 1, creamos y cargamos las tablas de la base de datos que usaremos en este ejercicio y lo hicimos en el tablespace creado en los apartados anteriores (al que solo tendrá acceso tu usuario). Conéctate al servidor a través de tu cliente favorito con tu usuario **GISXX**, y comprueba que todo está en orden.

Puedes lanzar las consultas con el cliente que prefieras. Cuando tengas la consulta bien creada, escríbela en un fichero de texto de forma que vayas creando un script "sql" con todas las respuestas.

Preguntas a responder

Se pide resolver en álgebra relacional y SQL una serie de consultas. Antes de ponerte a ello es importante que tengas en cuenta que tanto en AR como en SQL:

- Las cadenas de caracteres se encierran entre comillas simples.
- Los números no llevan separador de miles.

Las consultas son las siguientes:

1. Listado de empleados (con toda la información disponible) de los empleados que trabajan en el departamento cuyo identificador es 50.
2. Listado de empleados que reciben algún tipo de comisión. En el listado deben aparecer todos los datos de los empleados.
3. Listado con los nombres de los empleados que no tienen jefe.
4. Listado con los nombres de los empleados que trabajan en el departamento de ventas (`'Sales'`).
5. Listado con los nombres de los empleados que han trabajado en el departamento de ventas (`'Sales'`). PISTA: De eso nos informa la tabla `job_history`, que tiene el histórico de trabajos.
6. Listado con los nombres de los empleados que han trabajado en el departamento de ventas, pero que actualmente trabajan en otro departamento distinto.
7. Listado con los nombres de los empleados que trabajan en el departamento de informática (`'IT'`) que no trabajan en Europa (`region_name 'Europe'`), junto con el nombre del país en el que trabajan.
8. Nombres de los puestos que desempeñan los empleados en el departamento de informática, sin tuplas repetidas.
9. Nombres de los empleados que tienen personal a su cargo, es decir, que son jefes de algún empleado. Como es natural, sin repetición.
10. Listado de los nombres de los empleados que ganan más que su jefe, incluyendo también el nombre de su jefe y los salarios de ambos.
11. Listado de los nombres de los empleados que no trabajan en el mismo departamento que su jefe, junto con el nombre de su jefe y el departamento en el que trabajan ambos.

Entrega

La práctica se debe subir como un fichero de texto con el nombre `P2_GISXX.sql` (incluye para cada pregunta su consulta AR y su consulta SQL) en el campus virtual con los nombres de los dos miembros del grupo identificados al comienzo del fichero, y la pregunta que se está respondiendo indicada con un comentario.

NOTA: Los comentarios en los scripts SQL se indican con dos guiones seguidos (`--`).

La fecha límite de entrega será el **martes 15 de marzo a las 23:55h**.

Apéndice. Sintaxis del intérprete DES del álgebra relacional

Operators

This section includes descriptions for basic, additional and extended operators.

Basic operators

- Selection $\sigma_\theta(R)$. Select tuples in relation R matching condition θ.
  Concrete syntax:

  ```
  select Condition (Relation)
  ```

  Example:

  ```
  select a<>'a1' (c);
  ```

- Projection $\pi_{A1,...,An}(R)$. Return all tuples in R only with columns $A_1,...,A_n$.
  Concrete syntax:

  ```
  project A1,...,An (Relation)
  ```

  Example:

  ```
  project b (c);
  ```

  Note: Columns can be qualified when ambiguity arises, as in:

  ```
  project a.a (a product c)
  ```

  If no qualification is provided in presence of ambiguity, then a suitable column is arbitrarily chosen.

- Set union $R_1 \cup R_2$.
  Concrete syntax:

  ```
  Relation1 union Relation2
  ```

  Example:

  ```
  a union b;
  ```

- Set difference R_1 - R_2.
  Concrete syntax:

  ```
  Relation1 difference Relation2
  ```

  Example:

  ```
  a difference b;
  ```

- Cartesian product $R_1 \times R_2$.
  Concrete syntax:

  ```
  Relation1 product Relation2
  ```

  Example:

  ```
  a product b;
  ```

- Renaming $\rho_{R2(A1,...,An)}(R_1)$. Rename R_1 to R_2, and also arguments of R_1 to $A_1,...,A_n$.
  Concrete syntax:

```
rename Schema (Relation)
```

Example:

```
project v.b (rename v(b) (select true (a)));
```

Note:

The new name of a renamed relation must be different from the relation.

- Assignment $R_1(A1,...,An) \leftarrow R_2$. Create a new relation R_1 with argument names $A1,...,An$ as a copy of R_2. It allows defining new views.
  Concrete syntax:

```
Relation1 := Relation2
```

Example:

```
v(c) := select true (a);
```

Note:

Easy relation copying is supported by simply specifying relation names (no need to specify their arguments unless you want to change the destination argument names), as in:

```
u := v;      -- Same schema for u and v
u(b) := v;   -- Renamed schema for u w.r.t. v
```

Additional operators

These operators can be expressed in terms of basic operators, and include:

- Set intersection $R_1 \cap R_2$.
  Concrete syntax:

```
Relation1 intersect Relation2
```

Example:

```
a intersect b;
```

- Theta join $R_1 \bowtie_\theta R_2$. Equivalent to $\sigma_\theta(R_1 \times R_2)$.
  Concrete syntax:

```
Relation1 zjoin Condition Relation2
```

Example:

```
a zjoin a.a<b.b b;
```

- Natural (inner) join $R_1 \bowtie R_2$. Return tuples of R_1 joined with R_2 such that common attributes are pair wise equal and occur only once in the output relation.
  Concrete syntax:

```
Relation1 njoin Relation2
```

Example:

```
a njoin c;
```

- Division $R_1 \div R_2$. Return restrictions of tuples in R_1 to the attribute names of R_1 which are not in the schema of R_2, for which it holds that all their combinations with tuples in R_2 are present in R_1. The attributes in R_2 form a proper subset of attributes in R_1.
  Concrete syntax:

```
Relation1 division Relation2
```

Example:

```
a division c;
```

Extended operators

These operators *can not* be expressed in terms of former operators, and include:

- Extended projection (expressions and renamings) $\pi_{E1\ A1,\ldots,En\ An}(R)$. Return tuples of R with a new schema $R(A_1,\ldots,A_n)$ with columns $E_1,\ldots,E_n$ where each E_i is an expression built from constants, attributes of R, and built-in operators. If a given A_i is not provided, the name for the column is either the column E_i, if it is a column, or it is given an arbitrary new name.
  Concrete syntax:

```
project E1 A1,...,En An (Relation)
```

  Examples:

```
:-type(d(a:string,b:int)).
project b+1 (d);
project incb (project b+1 incb (d))
```

- Duplicate elimination $\delta(R)$. Return tuples in R, discarding duplicates.
  Concrete syntax:

```
distinct (Relation)
```

  Example:

```
distinct (project a (c));
```

  Note: As `distinct` is also a Datalog (meta)predicate, the query `distinct (c)` from the Datalog prompt would be solved as a Datalog query, instead of a RA one. Then, if you have to ensure your query will be evaluated by the RA processor, you can either switch to RA with `/ra`, or prepend the query with `/ra`, as follows:

```
DES> % Either switch to RA:
DES>/ra
DES-RA> distinct (project a (c));
DES> /datalog
DES> % Or simply add /ra
DES>/ra distinct (project a (c));
```

- Left outer join $R_1 \mathbin{⟕}_\theta R_2$. Includes all tuples of R_1 joined with matching tuples of R_2 w.r.t. condition θ. Those tuples of R_1 which do not have matching tuples of R_2 are also included in the result, and columns corresponding to R_2 are filled with null values.
  Concrete syntax:

```
Relation1 ljoin Condition Relation2
```

  Example:

```
a ljoin a=b b;
```

- Right outer join $R_1 \mathbin{⟖}_\theta R_2$. Equivalent to $R_2 \mathbin{⟕}_\theta R_1$. $R_1 \mathbin{⟕}_\theta R_2$
  Concrete syntax:

```
Relation1 rjoin Condition Relation2
```

Example:

```
a rjoin a=b b;
```

- Full outer join $R_1 \overset{\theta}{⟗} R_2$. Equivalent to $R_1 \overset{\theta}{⋈} R_2 \cup R_1 \overset{\theta}{⟕} R_2$.
  Concrete syntax:

  ```
  Relation1 fjoin Condition Relation2
  ```

  Example:

  ```
  a fjoin a=b b;
  ```

- Natural left outer join $R_1 ⟕ R_2$. Similar to left outer join but with no condition. Return tuples of R_1 joined with R_2 such that common attributes are pair wise equal and occur only once in the output relation.
  Concrete syntax:

  ```
  Relation1 nljoin Relation2
  ```

  Example:

  ```
  a nljoin c;
  ```

- Natural right outer join $R_1 ⟖ R_2$. Equivalent to $R_2 ⟕ R_1$.
  Concrete syntax:

  ```
  Relation1 nrjoin Relation2
  ```

  Example:

  ```
  a nrjoin c;
  ```

- Natural full outer join $R_1 ⟗ R_2$. Equivalent to $R_1 ⟕ R_2 \cup R_1 ⟖ R_2$.
  Concrete syntax:

  ```
  Relation1 nfjoin Relation2
  ```

  Example:

  ```
  a nfjoin c;
  ```

- Grouping with aggregations ${}_{G1,...,Gn}\varsigma^{E1,...,En}{}_{\theta}(R)$. Build groups of tuples in R so that: first, each tuple in the group have the same values for attributes $G_1,...,G_n$, second, matches condition θ (possibly including aggregate functions) and, third, is projected by expressions $E_1,...,E_n$ (also possibly including aggregate functions). An empty list of grouping attributes $G_1,...,G_n$ is denoted by an opening and a closing bracket (`[]`).
  Concrete syntax:

  ```
  group_by GroupingAtts ProjectingExprs HavingCond (Relation)
  ```

  Examples:

  ```
  % Number of employees
  group_by [] count(*) true (employee);
  % Employees with a salary greater than average salary,
  %   grouped by department
  group_by dept id salary > avg(salary) (employee);
  ```

- Sorting τ_L (R) Sort relation R with respect to sequence L [GUW02]. This sequence contains expressions which can be annotated by an ordering criteria, either **ascending** or **descending** (resp. abbreviated by **asc** and **desc**).
  Concrete syntax:

  ```
  sort Sequence (Relation)
  ```

  Examples:

  ```
  sort salary (employee);
  sort dept desc, name asc (employee);
  ```

- Top φ_N (R) Return the first N tuples of the relation R.
  Concrete syntax:

  ```
  top N (Relation)
  ```

  Example:

  ```
  top 10 (hits);
  ```