COMP9311 Week 05 Lecture

SQL Problem-solving

Steps in solving problems in SQL:

- know the schema, read the query request
- identify components of result tuples
- identify relevant data items and tables in schema
- build intermediate result tables (joins)
- combine intermediate tables to produce result
- compute values to appear in result tuples

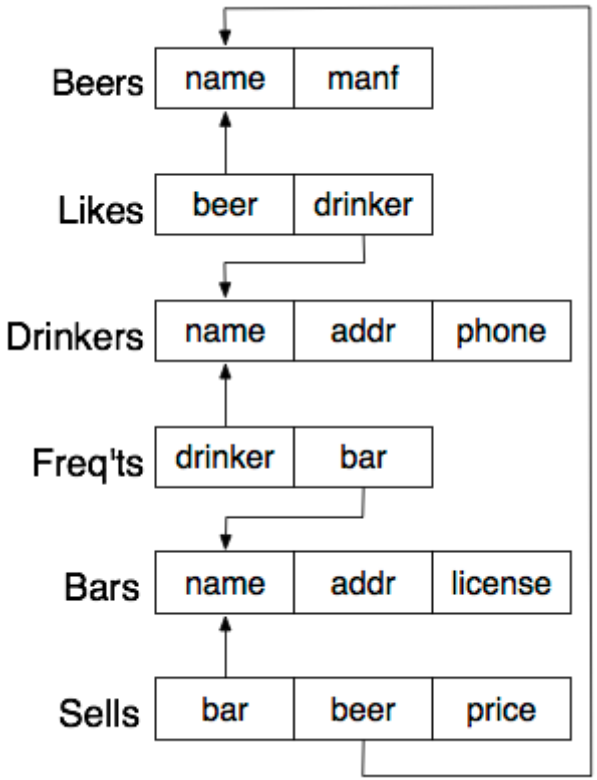
Design Elements:

- filters, joins (natural,inner,outer), sub-queries, groups, sets

[Join Examples]

Exercise: Queries on Beer Database

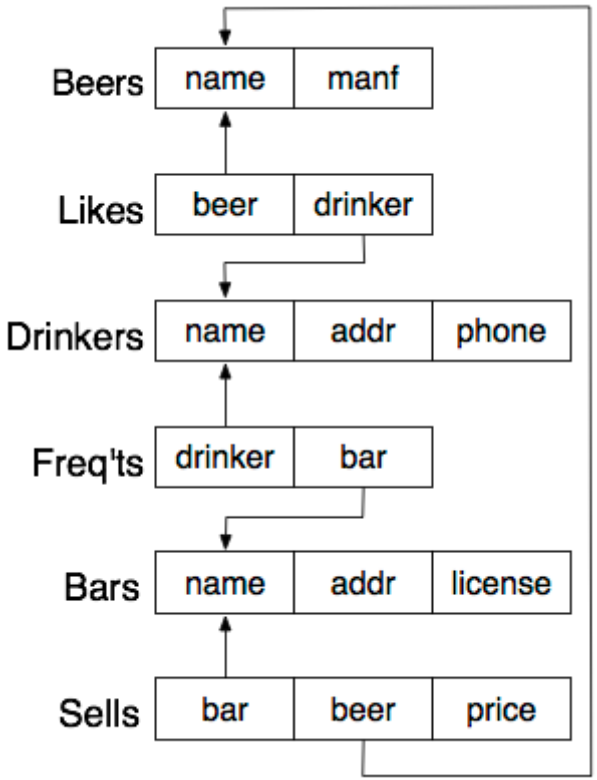
More queries on the Beer database:



9. How many beers does each brewer make?
10. Which brewer makes the most beers?
11. Bars where either Gernot or John drink.
12. Bars where both Gernot and John drink.
13. Find bars that serve New at the same price as the Coogee Bay Hotel charges for VB.
14. Find the average price of common beers (i.e. served in more than two hotels).
15. Which bar sells 'New' cheapest?

[Solutions]

... Exercise: Queries on Beer Database



More queries on the Beer database:

16. Which bar is most popular? (Most drinkers)
17. Which bar is most expensive? (Highest average price)
18. Which beers are sold at all bars?
19. Price of cheapest beer at each bar?
20. Name of cheapest beer at each bar?
21. How many drinkers are in each suburb?
22. How many bars in suburbs where drinkers live?
    (Must include suburbs with no bars)

[Solutions]

Stored Procedures

4/17

Stored procedures

- functions that are stored in DB along with data
- written in a language combining SQL and procedural ideas
- provide a way to extend operations available in database
- executed within the DBMS (close coupling with query engine)

Benefits of using stored procedures:

- minimal data transfer cost SQL ↔ procedural code
- user-defined functions can be nicely integrated with SQL

- procedures are managed like other DBMS data (ACID)
- procedures and the data they manipulate are held together

PostgreSQL Stored Procedures

PostgreSQL syntax for defining stored *functions*:

```
CREATE OR REPLACE FUNCTION
   funcName(arg1, arg2, ....) RETURNS retType
AS $$
String containing function definition
$$ LANGUAGE funcDefLanguage;
```

Notes:

- arg_i consists of *name type*
- `$$` ... `$$` are just another type of string quote
- function definition languages: SQL, PLpgSQL, Python, ...

Function Return Types

A PostgreSQL function can return a value which is

- `void` (i.e. no return value)
- an atomic data type (e.g. `integer, text, ...`)
- a tuple (e.g. table record type or tuple type)
- a set of atomic values (like a table column)
- a set of tuples (i.e. a table)

A function returning a set of values is similar to a view.

... Function Return Types

Examples of different function return types:

```
create function factorial(integer) returns integer ...
create function EmployeeOfMonth(date) returns Employee ...
create function allSalaries() returns setof float ...
create function OlderEmployees() returns setof Employee ...
```

Different kinds of functions are invoked in different ways:

```
select factorial();  -- returns one integer
select EmployeeOfMonth('2008-04-01');  -- returns (x,y,z)
select * from EmployeeOfMonth('2008-04-01'); -- one-row table
select * from allSalaries();  -- single-column table
select * from OlderEmployees();  -- subset of Employees
```

SQL Functions

PostgreSQL Manual: 35.4. Query Language (SQL) Functions

SQL Functions

9/17

PostgreSQL allows functions to be defined in SQL

```
CREATE OR REPLACE
   funcName(arg1type, arg2type, ....)
   RETURNS rettype
AS $$
   SQL statements
$$ LANGUAGE sql;
```

Within the function, arguments are accessed as `$1`, `$2`, ...

Return value: result of the last SQL statement.

rettype can be any PostgreSQL data type (incl tuples,tables).

Function returning a table: `returns setof` *TupleType*

... SQL Functions

10/17

Examples:

```
-- max price of specified beer
create or replace function
    maxPrice(text) returns float
as $$
select max(price) from Sells where beer = $1;
$$ language sql;

-- usage examples
select maxPrice('New');
 maxprice
----------
      2.8

select bar,price from sells
where beer='New' and price=maxPrice('New');
    bar     | price
------------+-------
 Marble Bar |   2.8
```

... SQL Functions

Examples:

```
-- set of Bars from specified suburb
create or replace function
    hotelsIn(text) returns setof Bars
as $$
select * from Bars where addr = $1;
$$ language sql;

-- usage examples
select * from hotelsIn('The Rocks');
      name       |   addr    | license
-----------------+-----------+---------
```

```
Australia Hotel | The Rocks |  123456
Lord Nelson     | The Rocks |  123888
```

PLpgSQL Functions

(PostgreSQL Manual: Chapter 39: PLpgSQL)

PLpgSQL

13/17

PLpgSQL = **P**rocedural **L**anguage extensions to **P**ost**g**re**SQL**

A PostgreSQL-specific language integrating features of:

- procedural programming and SQL programming

Provides a means for *extending DBMS functionality*, e.g.

- implementing constraint checking (triggered functions)
- complex query evaluation (e.g. recursive)
- complex computation of column values
- detailed control of displayed results

Defining PLpgSQL Functions

14/17

PLpgSQL functions are created (and inserted into db) via:

```
CREATE OR REPLACE
   funcName(param1, param2, ....)
   RETURNS rettype
AS $$
DECLARE
   variable declarations
BEGIN
   code for function
END;
$$ LANGUAGE plpgsql;
```

Note: the entire function body is a single SQL string.

PLpgSQL Function Parameters

Example: old-style function ("a","b") → "a'b"

```
CREATE OR REPLACE FUNCTION
    cat(text, text) RETURNS text
AS '
DECLARE
    x alias for $1;  -- alias for parameter
    y alias for $2;  -- alias for parameter
    result text;     -- local variable
BEGIN
    result := x||''''''''||y;
    return result;
END;
' LANGUAGE 'plpgsql';
```

Beware: never give aliases the same names as attributes.

... PLpgSQL Function Parameters

Example: new-style function ("a","b") → "a'b"

```
CREATE OR REPLACE FUNCTION
    add(x text, y text) RETURNS text
AS $add$
DECLARE
    result text;     -- local variable
BEGIN
    result := x||''''||y;
    return sum;
END;
$add$ LANGUAGE 'plpgsql';
```

Beware: never give parameters the same names as attributes.

One strategy: start all parameter names with an underscore.

Exercise: functions on (sets of) integers

Write PLpgSQL functions:

```
-- factorial n!
function fac(n integer) returns integer

-- returns integers 1..hi
function iota(hi integer) returns setof integer

-- returns integers lo..hi
function iota(lo integer, hi integer)
    returns setof integer

-- returns integers lo,lo+inc,..hi
function iota(lo integer, hi integer, inc integer)
    returns setof integer
```

[Solution]

Produced: 23 Aug 2016