SQL Recap and Introduction to PostgreSQL

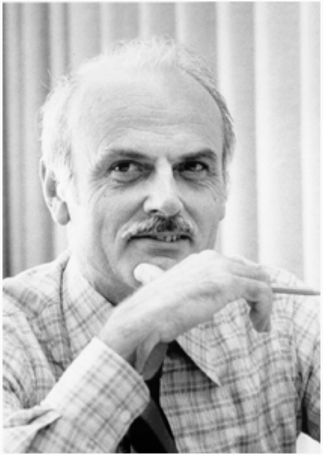
Sebastian Ernst, PhD

Course: Databases II, EAIiIBISIS.Ii8K.5dfa09851a120.22

Introduction

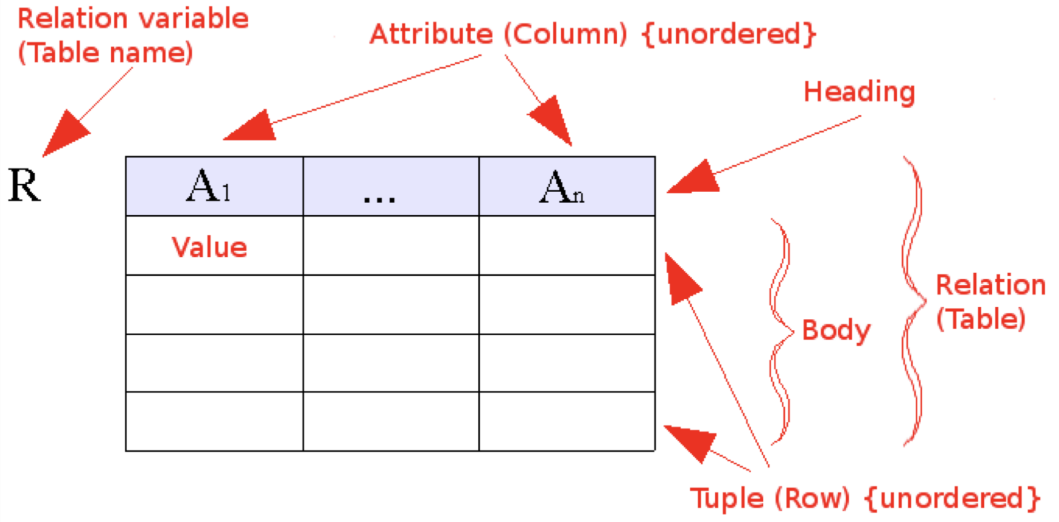
The relational data model

- Proposed in 1969 by Edgar F. Codd.
- Models data as a collection of predicates over a finite set of variables.
- The conceptual model includes entities, their attributes and relationships between them (one-to-one, one-to-many, many-to-many).



Source: Wikipedia

The relational data model



Source: Wikipedia

The SQL Language

About SQL

- SQL stands for Structured Query Language.
- Pronounced as ‘ess-que-ell’ or ‘sequel’.
- Performs many roles:
    - data modelling lanuage (DML),
    - data definition language (DDL),
    - data query language (DQL).

SQL: creating relations

Relations are created using the CREATE TABLE function:

```
CREATE TABLE weather (
    city      varchar(80),
    temp_lo   int,           -- low temperature
    temp_hi   int,           -- high temperature
    prcp      real,          -- precipitation
    date      date
);
```

To delete a table, use DROP TABLE:

```
DROP TABLE weather;
```

SQL: inserting data

Data is inserted using the INSERT statement:

```
INSERT INTO weather VALUES ('San Francisco', 46, 50, 0.25,
   '1994-11-27');
```

If the set or order of columns is different than the table structure, they need to be specified in the query:

```
INSERT INTO weather (city, temp_lo, temp_hi, prcp, date)
   VALUES ('San Francisco', 43, 57, 0.0, '1994-11-29');
```

SQL: basic operations

Selection and **projection** are two fundamental operations in SQL.

Selection chooses which columns are included in the result set of a SELECT query:

```
SELECT surname, age FROM employees;
SELECT * FROM employees;
```

Projection chooses which rows are returned by means of the WHERE clause:

```
SELECT * FROM employees WHERE age > 30;
```

SQL: aggregate functions

Aggregate functions combine a set of values into a single one. They include functions such as SUM, AVG, MIN, MAX, COUNT, etc.

```
SELECT AVG(age) FROM employees;
```

Aggregates may be computed for groups of rows instead of all records; these groups need to be explicitly defined.

```
SELECT AVG(age), department FROM employees GROUP BY department;
```

SQL: filtering in groups

The WHERE clause always applies to individual records (i.e. prior to aggregation). To perform selection on *groups*, one uses the HAVING clause.

```
SELECT AVG(age), department FROM employees GROUP BY department
   WHERE age > 30;     -- use only persons older than 30 for averages

SELECT AVG(age), department FROM employees GROUP BY department
   HAVING AVG(age) > 30; -- only show depts where average is over 30
```

SQL: inner joins

Joins merge **two sets of records**, creating a single result set.

Inner joins include only records which match both source sets:

```
SELECT * FROM weather, cities WHERE weather.city = cities.name;
SELECT * FROM weather INNER JOIN cities ON (weather.city = cities.name);
SELECT * FROM weather JOIN cities ON (weather.city = cities.name);
```

SQL: outer joins

Outer joins take all records from the left, right or both source sets:

```
SELECT * FROM weather LEFT OUTER JOIN cities ON (weather.city = cities.nam
SELECT * FROM weather LEFT JOIN cities ON (weather.city = cities.name);
```

SQL: JOIN syntax

The most general syntax (shown on the previous slide) uses the ON clause with any logical expression:

```
SELECT * FROM employees JOIN departments ON
   (employees.dept_id = departments.dept_id);
```

If matching is performed using the *equals* operator, and matching attributes have the same names in both data sets, the USING clause may be employed:

```
SELECT * FROM employees JOIN departments USING (dept_id);
```

In addition, if *all* attributes with identical names are to be matched, we have a *natural join*:

```
SELECT * FROM employees NATURAL JOIN departments;
```

SQL: set operations

Query results may be combined using the following operators:

- UNION – appends results of second query to those of the first one,
- INTERSECT – return all rows that are in both sets,
- EXCEPT – remove rows in the second result set from the first one.

All operators remove duplicates unless used with the ALL modifier.

SQL: subqueries

SQL queries may contain subqueries in various places – pay attention to the number of attributes and rows expected. Subqueries are often used with set operators:

- EXISTS – returns true if subquery has at least one row,
- IN/NOT IN – checks if value belongs/doesn't belong to set returned by subquery,
- ANY/SOME/ALL – similar to the above, but use arbitrary operators (instead of $=$ and $<>$). More info

Designing databases

Good practices for database design

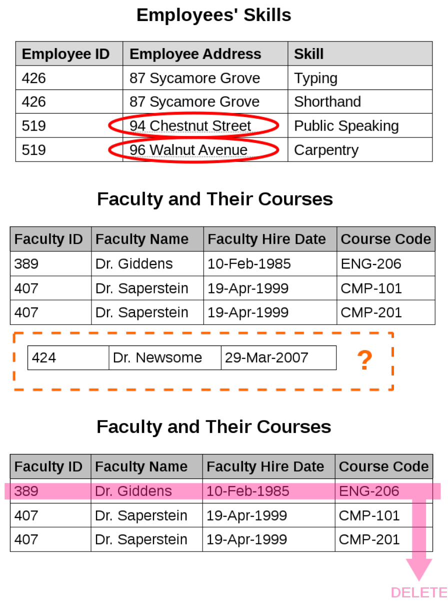
- Every real-world object = one entity = one relation.
- Every attribute occurs once, with its own object.
- Decompose non-atomic attributes.
- 1:n relationships: take primary key of “1”, migrate to “n” relation as foreign key.
- m:n relationships: create associative entity, use sum of primary keys from “m” and “n” as primary key.

Implementing „inheritance”

- The notion of *subclassing* is difficult to implement in a relational database, and leads to trade-offs.
- Possible approaches:
  1. one table for superclass (and common attributes), one table for each subclass.
  2. one table for each subclass; common attributes replicated in each one. Must use UNION to obtain a set on superclass level.
  3. one table for all; each row contains attributes of all subclasses (and those of the superclass). This leads to many NULLs.

Database normialization

- Rules to organise attributes and relations to avoid redundancy and improve integrity.
- Aimed at avoiding anomalies:
    - Update anomalies
    - Insertion anomalies
    - Deletion anomalies
- Normalisation provides a formal way to enforce good database design practices.



Employees' Skills

Employee ID	Employee Address	Skill
426	87 Sycamore Grove	Typing
426	87 Sycamore Grove	Shorthand
519	94 Chestnut Street	Public Speaking
519	96 Walnut Avenue	Carpentry

Faculty and Their Courses

Faculty ID	Faculty Name	Faculty Hire Date	Course Code
389	Dr. Giddens	10-Feb-1985	ENG-206
407	Dr. Saperstein	19-Apr-1999	CMP-101
407	Dr. Saperstein	19-Apr-1999	CMP-201

424	Dr. Newsome	29-Mar-2007	?

Faculty and Their Courses

Faculty ID	Faculty Name	Faculty Hire Date	Course Code
389	Dr. Giddens	10-Feb-1985	ENG-206
407	Dr. Saperstein	19-Apr-1999	CMP-101
407	Dr. Saperstein	19-Apr-1999	CMP-201

DELETE

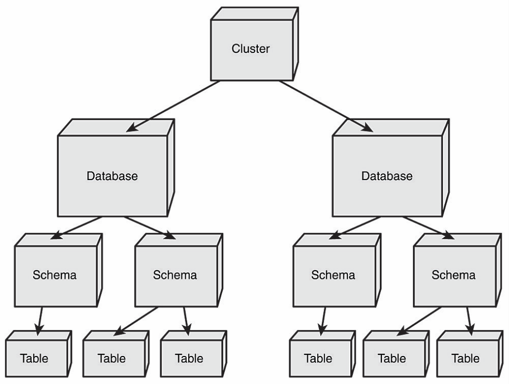
PostgreSQL

PostgreSQL basics

- PostgreSQL is a *relational database management system* (RDBMS)
- Cross-platform & open-source
- Client-server architecture:
    - server listens on 5432/tcp by default
    - clients use *libpq* to connect

PostgreSQL concepts

- A *cluster* is one or more database instances.
- Each *database* contains at least one schema, public.
- Any number of schemas can be created.
- Tables and other objects are assigned to a *specific schema*.



{source: Korry Douglas, PostgreSQL, 2nd Edition}

PostgreSQL schemas

- Creating a schema:

  ```
  CREATE SCHEMA my_other_schema;
  ```

- Objects in schemas can be referred to by schema.object.
- Otherwise the DB looks in schemas listed in `search_path`, which act much like the `PATH` system environment variable.
- To set search_path:

  ```
  SET search_path TO public, my_other_schema;
  SET search_path TO my_other_schema, public;
  SET search_path TO my_other_schema;
  ```

PostgreSQL clients

- Collection of command-line utilities, including psql and several helper programs.
- Any application can use libpq, usually via some wrapper.
- GUI apps: phppgadmin, Adminer, pgAdmin

Using *psql*: command-line switches

```
psql [option...] [dbname [username]]
```

Useful switches:

- -d dbname – specify database name
- -h hostname – specify host to connect to; if omitted, it will connect to local UNIX domain socket specified in the config
- -U username – specify username
- -l – list databases
- -c command – run specified command and exit
- -f filename – run commands from file and exit

Using *psql*: interactive shell

Some useful meta-commands:

- `\l` – list databases
- `\d` – list objects
- `\dE`, `\di`, `\dm`, `\ds`, `\dt`, `\dv` – list objects of specific type: foreign table, index, materialized view, sequence, table, and view
- `\i` – read commands from file
- `\q` – quit

More: psql documentation