Introduction to Relational Databases Part 1: Why?

Claude Rubinson
University of Houston—Downtown

rubinsonc@uhd.edu
cjr@grundrisse.org

September 21, 2011

What Makes a Database Relational?

- Based on relational algebra (set-theory)
- ACID properties
  - Atomicity
    - Transactions are “all or nothing”
  - Consistency
    - Database is always in a consistent state
  - Isolation
    - One transaction doesn't affect another
  - Durability
    - Transactions persist across system crashes

Relational Databases versus Conventional Datasets

- Many tables rather than one
- Rows are unordered
- Columns are unordered
- Relationships among tables (database schema) represent the structure of the data

Why Use a Relational Database?

Conventional answer (for organizations):

- Scalability, reliability, industry-standard

For individuals:

- Supports post-hoc/ad-hoc queries
- Allows (encourages, forces) you to accurately model the domain of interest

Some Terminology

- Tables (Relations, Relvars)
- Rows (Tuples)
- Columns (Attributes)
- Primary Keys

Programmers

ProgUID	LastName	FirstName
1	Ritchie	Dennis
2	Stallman	Richard
3	Torvalds	Linus
6	Wall	Larry

Relational Database Design

- Normalization—the process of eliminating redundancy
- Functional dependencies
  - “If I know one attribute, I can determine another”
  - Singular dependencies: A -> B
  - Multivalued dependencies: A ->> B
  - Defines the structure of the data

Normalization

- The process of eliminating redundancy
- Done wrong, the database will be difficult to maintain and information will be difficult or impossible to retrieve. Even worse, incorrect information may be retrieved.

Conventional Database

Class	Teacher	Student
Econ 101	Smith	John
Econ 101	Smith	Mary
Econ 101	Smith	Jane
Econ 201	Smith	Jane
Art Hist 101	Jones	Mary
Art Hist 101	Jones	Smith

Normalized Database

People

PersonUID	Person	Age
1	Smith	46
2	Jones	38
3	John	22
4	Mary	18
5	Jane	24
6	James	24

Normalized Database

People

PersonUID	Person	Age
1	Smith	46
2	Jones	38
3	John	22
4	Mary	18
5	Jane	24
6	James	24

Normalized Database

People

PersonUID	Person	Age
1	Smith	46
2	Jones	38
3	John	22
4	Mary	18
5	Jane	24
6	James	24

Classes

ClassUID	Class
1	Econ 101
2	Econ 201
3	Art Hist 101
4	Soc 101

Normalized Database

People

PersonUID	Person	Age
1	Smith	46
2	Jones	38
3	John	22
4	Mary	18
5	Jane	24
6	James	24

ClassTeacher

ClassUID	PersonUID
1	1
2	1
3	2

Classes

ClassUID	Class
1	Econ 101
2	Econ 201
3	Art Hist 101
4	Soc 101

Normalized Database

People

PersonUID	Person	Age
1	Smith	46
2	Jones	38
3	John	22
4	Mary	18
5	Jane	24
6	James	24

ClassTeacher

ClassUID	PersonUID
1	1
2	1
3	2

ClassStudents

ClassUID	PersonUID
1	3
1	4
1	5
2	5
3	4
3	1

Classes

ClassUID	Class
1	Econ 101
2	Econ 201
3	Art Hist 101
4	Soc 101

What is SQL?

- Structured Query Language
- Pronounced “Ess Que El” or “Sequel”
- Standardized, English-like language for interacting with Relational Database Management Systems (RDBMS)
- Set (technically, “Bag”) based
- Declarative (non-procedural) language
- But, incompatible proprietary extensions

Query Language

```
SELECT * FROM People;
```

Query Language

```
SELECT * FROM People;

personuid | person | age
----------+--------+----
        1 | Smith  | 46
        2 | Jones  | 38
        3 | John   | 22
        4 | Mary   | 18
        5 | Jane   | 24
        6 | James  | 24
(6 rows)
```

Query Language

```
SELECT person FROM People
WHERE age >= 20
ORDER BY person;
```

Query Language

```
SELECT person FROM People
WHERE age >= 20
ORDER BY person;

 person
--------
 James
 Jane
 John
 Jones
 Smith
(5 rows)
```

Query Language

```
SELECT avg(age) as avg_age,
FROM People
WHERE age >= 20;
```

Query Language

```
SELECT avg(age) as avg_age,
FROM People
WHERE age >= 20;

       avg_age
---------------------
 30.8000000000000000
(1 row)
```

Query Language

```
SELECT Classes.class, count(*) as N
FROM Classes, ClassStudents as CS
WHERE Classes.classuid=CS.classuid
GROUP BY Classes.class
HAVING count(CS.personuid) >= 2;
```

Query Language

```
SELECT Classes.class, count(*) as N
FROM Classes, ClassStudents as CS
WHERE Classes.classuid=CS.classuid
GROUP BY Classes.class
HAVING count(CS.personuid) >= 2;

     class     | n
---------------+---
 Art Hist 101 | 2
 Econ 101      | 3
(2 rows)
```

Query Language

```
     class     | n | teacher
---------------+---+---------
 Art Hist 101 | 2 | Jones
 Econ 101     | 3 | Smith
(2 rows)
```

Query Language

```
SELECT Classes.class, count(*) as N

FROM Classes, ClassStudents as CS

WHERE Classes.classuid = CS.classuid


GROUP BY Classes.class
HAVING count(CS.personuid) >= 2;

     class     | n | teacher
---------------+---+---------
 Art Hist 101  | 2 | Jones
 Econ 101      | 3 | Smith
(2 rows)
```

Query Language

```
SELECT Classes.class, count(*) as N

FROM Classes, ClassStudents as CS,
  ClassTeacher as CT, People
WHERE Classes.classuid = CS.classuid AND
  Classes.classuid = CT.classuid AND
  CT.personuid = People.personuid
GROUP BY Classes.class
HAVING count(CS.personuid) >= 2;

     class     | n | teacher
---------------+---+----------
 Art Hist 101 | 2 | Jones
 Econ 101      | 3 | Smith
(2 rows)
```

Query Language

```
SELECT Classes.class, count(*) as N,
  People.person as Teacher
FROM Classes, ClassStudents as CS,
  ClassTeacher as CT, People
WHERE Classes.classuid = CS.classuid AND
  Classes.classuid = CT.classuid AND
  CT.personuid = People.personuid
GROUP BY Classes.class, People.person
HAVING count(CS.personuid) >= 2;

     class     | n | teacher
---------------+---+---------
 Art Hist 101  | 2 | Jones
 Econ 101      | 3 | Smith
(2 rows)
```

Types of Joins

- Cross Join (Cartesian Product)

```
SELECT *
FROM table1, table2;
```

- Inner Join

```
SELECT *
FROM table1, table2
WHERE table1.joincol=table2.joincol;
```

```
SELECT *
FROM table1 INNER JOIN table2
    ON (table1.joincol=table2.joincol);
```

Types of Joins

- Outer Joins Create NULLs

```
SELECT *
FROM table1 LEFT JOIN table2
    ON (table1.joincol=table2.joincol);

SELECT *
FROM table1 RIGHT JOIN table2
    ON (table1.joincol=table2.joincol);

SELECT *
FROM table2 FULL JOIN table2
    ON (table1.joincol=table2.joincol);
```

Review of RDBMSes

- Oracle, MS SQL Server
  - Industry standards
  - Expensive
- MS Access, LibreOffice Base
  - Graphical
  - “User friendly”
  - Inexpensive
  - Slow/Not scalable
  - LO Base can act as frontend for MySQL, PostgreSQL

Review of RDBMSes

- MySQL
  - Open-source
  - Fast
  - Lots of newbie friendly documentation
- PostgreSQL
  - Open-source
  - Strict(er) adherence to relational model
  - High signal:noise ratio on mailing lists, discussion groups, etc.
  - Very thorough documentation

Recommended Resources

- *SQL for Smarties* by Joe Celko
- *Database Modeling & Design* by Toby J. Teory
- PostgreSQL Online Documentation at http://www.postgresql.org/docs/
- *An Introduction to Database Systems* by Chris Date
- *SQL and Relational Theory* by Chris Date
- *Developing Time-Oriented Database Applications in SQL* by Richard T. Snodgrass