Database Design and Programming

Peter Schneider-Kamp

DM 505, Spring 2009, 3rd Quarter

Course Organisation

- Literature
  - *Database Systems: The Complete Book*
- Evaluation
  - Project and 1-day take-home exam, 7 scale
- Project
  - Design and implementation of a database using PostgreSQL and JDBC
- Schedule
  - 4/2 lectures a week, 2/4 exercises a week

Course Organisation

- Literature
  - *Database Systems: The Complete Book*
  - Book has not arrived at the book store yet ☹
  - Chapters 1 & 2 available online
  - Chapter 5.1 as copies
  - “drop ship” from the US (January 29)

(Preliminary) Course Schedule

Week	Room	06	07	08	09	10	11	12
Mon 12-14	U9	L	L	L	L	L	L	L
Wed 10-12	U9	E	E	L	E	E	E	E
Thu 10-12	(U9)	L	E (U148)	E	E	L	E	L

- 4/2 lectures, 2/4 exercises
- Lecture and exercise swapped in Week 8
- always U9 except for 1 exercise in U148

Where are Databases used?

It used to be about boring stuff:

- Corporate data
  - payrolls, inventory, sales, customers, accounting, documents, ...
- Banking systems
- Stock exchanges
- Airline systems
- ...

Where are Databases used?

Today, databases are used in all fields:

- Web backends:
  - Web search (Google, Live, Yahoo, ...)
  - Social networks (Facebook, ...)
  - Blogs, discussion forums
  - ...
- Integrating data (data warehouses)
- Scientific and medical databases
- ...

Why are Databases used?

- Easy to use
- Flexible searching
- Efficiency
- Centralized storage, multi-user access
- Scalability (large amounts of data)
- Security and consistency
- Abstraction (implementation hiding)
- Good data modeling

Why learn about Databases?

- Very widely used
- Part of most current software solutions
- DB expertise is a career asset
- Interesting:
  - Mix of different requirements
  - Mix of different methodologies
  - Integral part of data driven development
  - Interesting real word applications

Short History of Databases

- Early 60s: *Integrated Data Store*, General Electric, first DBMS, network data model
- Late 60s: *Information Management System*, IBM, hierarchical data model
- 1970: E. Codd: Relational data model, relational query languages, Turing prize
- Mid 70s: First relational DBMSs (IBM System R, UC Berkeley Ingres, ...)
- 80s: Relational model de facto standard

Short History of Databases

- 1986: SQL standardized
- 90s: Object-relational databases, object-oriented databases
- Late 90s: XML databases
- 1999: SQL incorporates some OO features
- 2003, 2006: SQL incorporates support for XML data
- …

Current Database Systems

- DBMS = Database Management System
- Many vendors (Oracle, IBM DB2, MS SQL Server, MySQL, PostgreSQL, . . .)
- All rather similar
- Very big systems, but easy to use
- Common features:
  - Relational model
  - SQL as the query language
  - Server-client architecture

Transactions

- Groups of statements that need to be executed together
- Example:
  - Transferring money between accounts
  - Need to subtract amount from 1^{st} account
  - Need to add amount to 2^{nd} account
  - Money must not be lost!
  - Money should not be created!

ACID

Required properties for transactions

- “A“ for “atomicity“ – all or nothing of transactions
- “C“ for “consistency“ – constraints hold before and after each transaction
- “I“ for “isolation“ – illusion of sequential execution of each transaction
- “D“ for “durability“ – effect of a completed transaction may not get lost

Database Develolpment

- Requirement specification (not here)
- Data modeling
- Database modeling
- Application programming
- Database tuning

Database Course Contents

- E/R-model for data modeling
- Relational data model
- SQL language
- Application programming (JDBC)
- Basic implementation principles
- DB tuning

Note: DM 505 ≠ SQL course

DM 505 ≠ PostgreSQL course

Data Model

What is a Data Model?

1. Mathematical representation of data
   - relational model = tables
   - semistructured model = trees/graphs
   - ...
2. Operations on data
3. Constraints

A Relation is a Table

Attributes (column headers)

Tuples (rows)

name	manf
Odense Classic	Albani
Erdinger Weißbier	Erdinger

Beers

Relation name

Note: Order of attributes and rows is irrelevant (sets / bags)

Schemas

- *Relation schema* =
  relation name and attribute list
  - Optionally: types of attributes
  - Example: Beers(name, manf) or Beers(name: string, manf: string)
- *Database* = collection of relations
- *Database schema* = set of all relation schemas in the database

Why Relations?

- Very simple model
- *Often* matches how we think about data
- Abstract model that underlies SQL,
  the most important database language
  today

Our Running Example

Beers(name, manf)

Bars(name, addr, license)

Drinkers(name, addr, phone)

Likes(drinker, beer)

Sells(bar, beer, price)

Frequents(drinker, bar)

- Underline = *key* (tuples cannot have the same value in all key attributes)
  - Excellent example of a constraint

Database Schemas in SQL

- SQL is primarily a query language, for getting information from a database
- But SQL also includes a *data-definition* component for describing database schemas

Creating (Declaring) a Relation

- Simplest form is:

```
CREATE TABLE <name> (
    <list of elements>
);
```

- To delete a relation:

```
DROP TABLE <name>;
```

Elements of Table Declarations

- Most basic element:
  an attribute and its type
- The most common types are:
  - INT or INTEGER (synonyms)
  - REAL or FLOAT (synonyms)
  - CHAR(*n*) = fixed-length string of *n* characters
  - VARCHAR(*n*) = variable-length string of up to *n* characters

Example: Create Table

```
CREATE TABLE Sells (
     bar     CHAR(20),
     beer    VARCHAR(20),
     price   REAL
  );
```

SQL Values

- Integers and reals are represented as you would expect
- Strings are too, except they require single quotes
  - Two single quotes = real quote, e.g., `'Trader Joe''s Hofbrau Bock'`
- Any value can be NULL
  - (like Objects in Java)

Dates and Times

- DATE and TIME are types in SQL
- The form of a date value is:

  DATE ’yyyy-mm-dd’

  - Example: `DATE ’2009-02-04’` for February 4, 2009

Times as Values

- The form of a time value is:

  TIME 'hh:mm:ss'

  with an optional decimal point and fractions of a second following
  - Example: `TIME '15:30:02.5'` = two and a half seconds after 15:30

Declaring Keys

- An attribute or list of attributes may be declared PRIMARY KEY or UNIQUE
- Either says that no two tuples of the relation may agree in all the attribute(s) on the list
- There are a few distinctions to be mentioned later

Declaring Single-Attribute Keys

- Place PRIMARY KEY or UNIQUE after the type in the declaration of the attribute
- Example:

```
CREATE TABLE Beers (
        name    CHAR(20) UNIQUE,
        manf    CHAR(20)
   );
```

Declaring Multiattribute Keys

- A key declaration can also be another element in the list of elements of a CREATE TABLE statement
- This form is essential if the key consists of more than one attribute
  - May be used even for one-attribute keys

Example: Multiattribute Key

- The bar and beer together are the key for Sells:

```
CREATE TABLE Sells (
        bar      CHAR(20),
        beer     VARCHAR(20),
        price    REAL,
        PRIMARY KEY (bar, beer)
  );
```

PRIMARY KEY vs. UNIQUE

1. There can be only one PRIMARY KEY for a relation, but several UNIQUE attributes
2. No attribute of a PRIMARY KEY can ever be NULL in any tuple. But attributes declared UNIQUE may have NULL's, and there may be several tuples with NULL

Changing a Relation Schema

- To delete an attribute:

  ALTER TABLE <name> DROP <attribute>;

- To add an attribute:

  ALTER TABLE <name> ADD <element>;

- Examples:

  ALTER TABLE Beers ADD prize CHAR(10);

  ALTER TABLE Drinkers DROP phone;

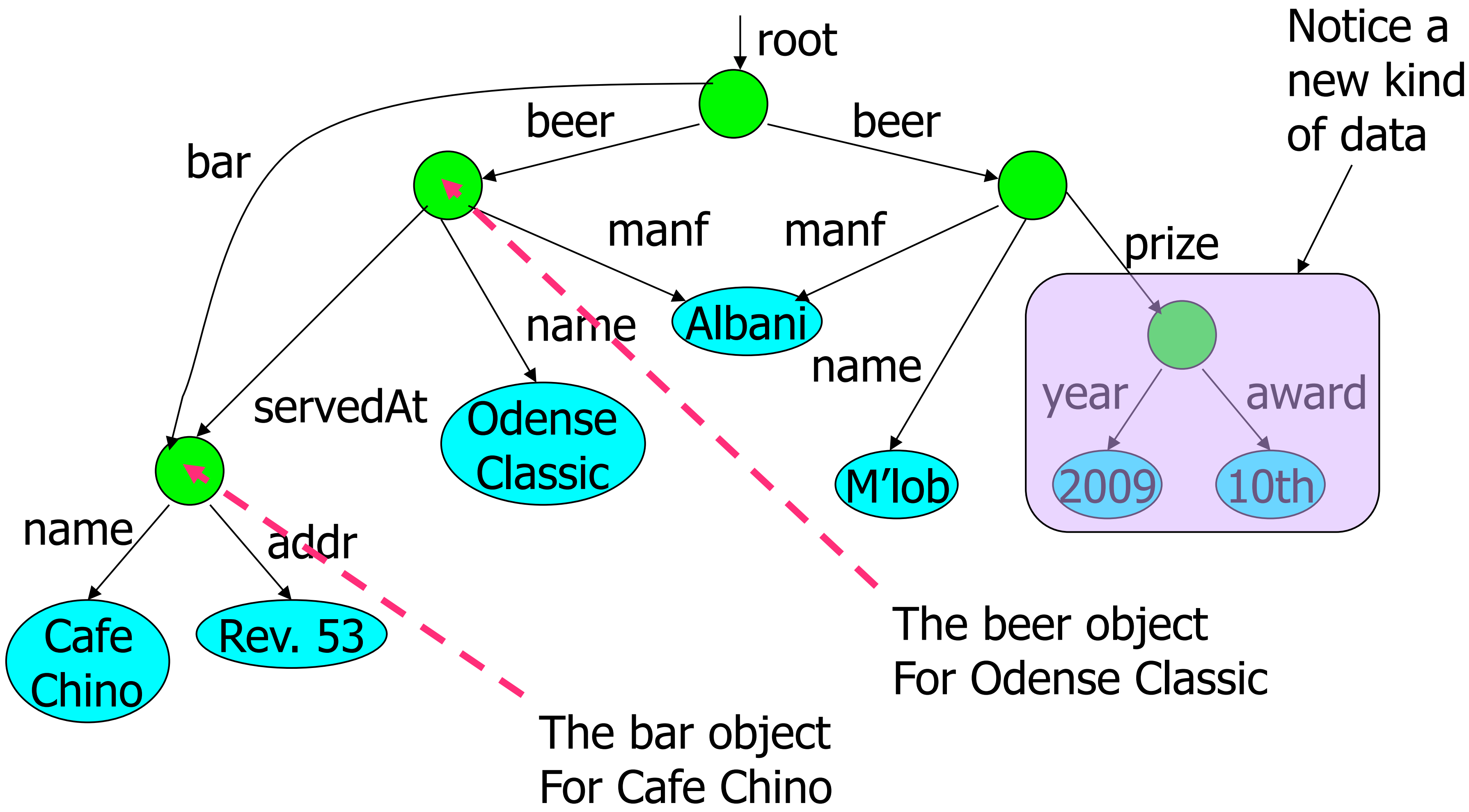
Semistructured Data

- Another data model, based on trees
- Motivation: flexible representation of data
- Motivation: sharing of *documents* among systems and databases

Graphs of Semistructured Data

- Nodes = objects
- Labels on arcs (like attribute names)
- Atomic values at leaf nodes (nodes with no arcs out)
- Flexibility: no restriction on:
  - Labels out of a node
  - Number of successors with a given label

Example: Data Graph



XML

- XML = *Extensible Markup Language*
- While HTML uses tags for formatting (e.g., "italic"), XML uses tags for semantics (e.g., "this is an address")
- Key idea: create tag sets for a domain (e.g., genomics), and translate all data into properly tagged XML documents

XML Documents

- Start the document with a *declaration*, surrounded by <?xml … ?>
- Typical:

```
<?xml version = “1.0” encoding
  = “utf-8” ?>
```

- Document consists of one *root tag* surrounding nested tags

Tags

- Tags, as in HTML, are normally matched pairs, as <FOO> … </FOO>
  - Optional single tag <FOO/>
- Tags may be nested arbitrarily
- XML tags are case sensitive

Example: an XML Document

```
<?xml version = "1.0" encoding = "utf-8" ?>
<BARS>
   <BAR><NAME>Cafe Chino</NAME>
      <BEER><NAME>Odense Classic</NAME>
          <PRICE>20</PRICE></BEER>
      <BEER><NAME>Erdinger Weißbier</NAME>
          <PRICE>35</PRICE></BEER>
   </BAR>
   <BAR> …
</BARS>
```

A NAME subobject

A BEER subobject

Attributes

- Like HTML, the opening tag in XML can have attribute = value pairs
- Attributes also allow linking among elements (discussed later)

Bars, Using Attributes

```
<?xml version = “1.0” encoding = “utf-8” ?>
<BARS>
   <BAR name = “Cafe Chino”>
      <BEER name = “Odense Classic” price = 20 />
      <BEER name = “Erdinger Weißbier” price = 35 />
   </BAR>
   <BAR> …
</BARS>
```

name and price are attributes

Notice Beer elements have only opening tags with attributes.

DTD’s (Document Type Definitions)

- A grammatical notation for describing allowed use of tags.
- Definition form:

<!DOCTYPE <root tag> [

<!ELEMENT <name>(<components>)>

. . . more elements . . .

]>

Example: DTD

```
<!DOCTYPE BARS [
  <!ELEMENT BARS (BAR*)>
  <!ELEMENT BAR (NAME, BEER+)>
  <!ELEMENT NAME (#PCDATA)>
  <!ELEMENT BEER (NAME, PRICE)>
  <!ELEMENT PRICE (#PCDATA)>
]>
```

A BARS object has zero or more BAR's nested within.

A BAR has one NAME and one or more BEER subobjects.

A BEER has a NAME and a PRICE.

NAME and PRICE are HTML text.

Attributes

- Opening tags in XML can have *attributes*
- In a DTD,

`<!ATTLIST` *E* `...>`

declares an attribute for element *E*, along with its datatype

Example: Attributes

```
<!ELEMENT BEER EMPTY>
  <!ATTLIST name CDATA #REQUIRED,
     manf CDATA #IMPLIED>
```

No closing tag or subelements

Character string

Required = “must occur”; Implied = “optional

Example use:
<BEER name=“Odense Classic” />

Summary 1

Things you should know now:

- Basic ideas about databases and DBMSs
- What is a data model?
- Idea and Details of the relational model
- SQL as a data definition language

Things given as background:

- History of database systems
- Semistructured data model

Relational Algebra

What is an “Algebra”

- Mathematical system consisting of:
  - *Operands* – variables or values from which new values can be constructed
  - *Operators* – symbols denoting procedures that construct new values from given values
- Example:
  - Integers ..., -1, 0, 1, ... as operands
  - Arithmetic operations +/- as operators

What is Relational Algebra?

- An algebra whose operands are relations or variables that represent relations
- Operators are designed to do the most common things that we need to do with relations in a database
  - The result is an algebra that can be used as a *query language* for relations

Core Relational Algebra

- Union, intersection, and difference
  - Usual set operations, but *both operands must have the same relation schema*
- Selection: picking certain rows
- Projection: picking certain columns
- Products and joins: compositions of relations
- Renaming of relations and attributes

Selection

- $R_1 := \sigma_C(R_2)$
  - C is a condition (as in "if" statements) that refers to attributes of R_2
  - R_1 is all those tuples of R_2 that satisfy C

Example: Selection

Relation Sells:

bar	beer	price
Cafe Chino	Od. Cla.	20
Cafe Chino	Erd. Wei.	35
Cafe Bio	Od. Cla.	20
Bryggeriet	Pilsener	31

ChinoMenu := $\sigma_{bar=\text{“Cafe Chino”}}$(Sells):

bar	beer	price
Cafe Chino	Od. Cla.	20
Cafe Chino	Erd. Wei.	35

Projection

- $R_1 := \pi_L (R_2)$
  - *L* is a list of attributes from the schema of R_2
  - R_1 is constructed by looking at each tuple of R_2, extracting the attributes on list *L*, in the order specified, and creating from those components a tuple for R_1
  - Eliminate duplicate tuples, if any

Example: Projection

Relation Sells:

bar	beer	price
Cafe Chino	Od. Cla.	20
Cafe Chino	Erd. Wei.	35
Cafe Bio	Od. Cla.	20
Bryggeriet	Pilsener	31

Prices := $\pi_{beer,price}$(Sells):

beer	price
Od. Cla.	20
Erd. Wei.	35
Pilsener	31

Extended Projection

- Using the same π_L operator, we allow the list L to contain arbitrary expressions involving attributes:
  1. Arithmetic on attributes, e.g., $A+B$->C
  2. Duplicate occurrences of the same attribute

Example: Extended Projection

R = (

A	B
1	2
3	4

)

$\pi_{A+B\text{->}C,A,A}$ (R) =

C	A_1	A_2
3	1	1
7	3	3

Product

- $R_3 := R_1 \times R_2$
  - Pair each tuple t_1 of R_1 with each tuple t_2 of R_2
  - Concatenation t_1t_2 is a tuple of R_3
  - Schema of R_3 is the attributes of R_1 and then R_2, in order
  - But beware attribute *A* of the same name in R_1 and R_2: use $R_1.A$ and $R_2.A$

Example: $R_3 := R_1 \text{ X } R_2$

$R_1($

A,	B
1	2
3	4

$)$

$R_2($

B,	C
5	6
7	8
9	10

$)$

$R_3($

A,	R_1.B,	R_2.B,	C
1	2	5	6
1	2	7	8
1	2	9	10
3	4	5	6
3	4	7	8
3	4	9	10

$)$

Theta-Join

- $R_3 := R_1 \bowtie_C R_2$
  - Take the product R_1 X R_2
  - Then apply σ_C to the result
- As for σ, C can be any boolean-valued condition
  - Historic versions of this operator allowed only A θ B, where θ is =, <, etc.; hence the name "theta-join"

Example: Theta Join

Sells(

bar,	beer,	price
C.Ch.	Od.C.	20
C.Ch.	Er.W.	35
C.Bi.	Od.C.	20
Bryg.	Pils.	31

)

Bars(

name,	addr
C.Ch.	Reventlo.
C.Bi.	Brandts
Bryg.	Flakhaven

)

BarInfo := Sells $\bowtie_{\text{Sells.bar = Bars.name}}$ Bars

BarInfo(

bar,	beer,	price,	name,	addr
C.Ch.	Od.C.	20	C.Ch.	Reventlo.
C.Ch.	Er.W.	35	C.Ch.	Reventlo.
C.Bi.	Od.C.	20	C.Bi.	Brandts
Bryg.	Pils.	31	Bryg.	Flakhaven

)

Natural Join

- A useful join variant (*natural* join) connects two relations by:
  - Equating attributes of the same name, and
  - Projecting out one copy of each pair of equated attributes
- Denoted $R_3 := R_1 \bowtie R_2$

Example: Natural Join

Sells(

bar,	beer,	price
C.Ch.	Od.Cl.	20
C.Ch.	Er.We.	35
C.Bi.	Od.Cl.	20
Bryg.	Pils.	31

)

Bars(

bar,	addr
C.Ch.	Reventlo.
C.Bi.	Brandts
Bryg.	Flakhaven

)

BarInfo := Sells $\bowtie$ Bars

Note: Bars.name has become Bars.bar to make the natural join "work"

BarInfo(

bar,	beer,	price,	addr
C.Ch.	Od.Cl.	20	Reventlo.
C.Ch.	Er.We.	35	Reventlo.
C.Bi.	Od.Cl.	20	Brandts
Bryg.	Pils.	31	Flakhaven

)