CS 5142
Scripting Languages

10/16/2015

Web Applications

Databases

Outline

- Databases

Soap-box

Why use a Database?

- Store data for months or years
  - Database may live longer than the web application(s) that you write for it
- Why not just use simple ad-hoc files?
  - Database remains consistent in the presence of multiple concurrent accesses
  - Database scales better when there is a lot of data, or a lot of accesses
  - Don't reinvent the wheel

Concepts

Relational Databases

- Database = collection of tables
- Table = relation = set of rows w/ same columns
- Row = tuple = one value per column
- Column = attribute = name+primitive type
- Only store primitive values, never nest tables

Recipe

name	serves
cake	4
gravy	4

Ingredient

name	taste
flour	bland
salt	salty

RecIng

rec	ing	qty	unit
cake	flour	2.5	cup
cake	milk	3	tbsp
gravy	salt	2	tsp
gravy	sugar	10	g

Cuisine

name	continent
Italian	Europe
Indian	Asia

RecCui

rec	cui
daal	Indian
pizza	Italian

Relational Algebra

- *Union*: Elements of R and elements of S
- *Difference*: Elements of R not in S
- *Cross Product*: Set of pairs (r,s), such that r in R and s in S
- *Selection* (σ): subset of the rows
- *Projection* (π): subset of column attributes
- *Natural Join*: set of all combinations of tuples with common attributes

Soap-box

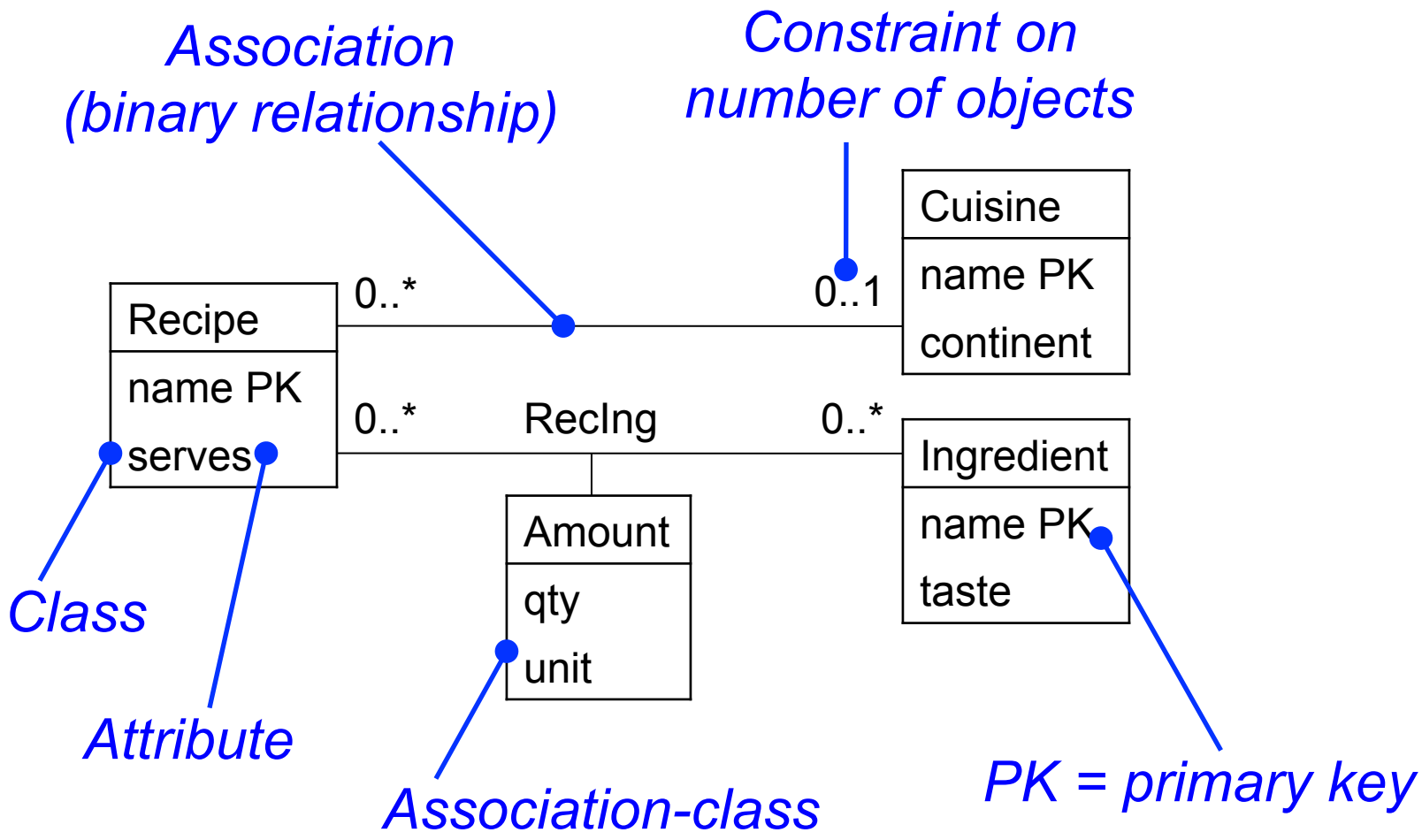
Non-Relational Data Stores

- Key-Value store:
  - *Cassandra*: distributed, tunable-consistency
  - *Memchached*: in-memory, used by Twitter, Facebook
- Document database:
  - *CouchDB:* JSON to store data, query with JavaScript
  - *MongoDB*
- Object-oriented databases:
  - *Hibernate*: ORM
  - *EJB*: Managed container (persistence + more)

From UML to Relations

- For each class, create a relation
  - Relation name = class name
  - Relation columns = class attributes
- For each association, create a relation
  - Relation name = combine class names
  - Relation columns = primary keys of both classes, plus attributes of association class, if any

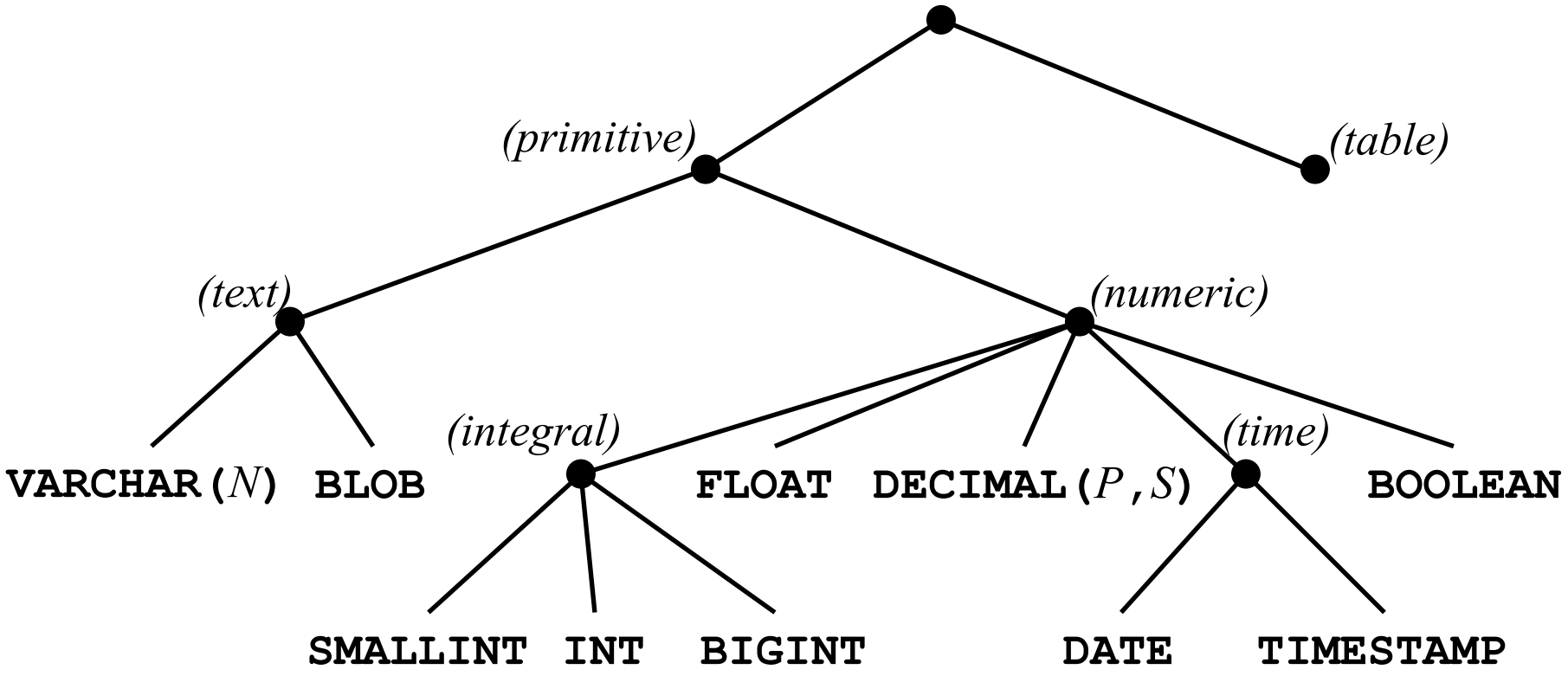
UML with Associations



About SQL

- Structured Query Language
  - Query information from relational database
  - Declarative: describe what information to find, not how to find it
- SQL consists of two parts
  - DDL = Data Definition Language
  - DML = Data Manipulation Language
- Each database product (sqlite, MySQL, Oracle, DB2, …) has own SQL dialect
  - We use sqlite in this course

Types



Most database products have additional primitive types.

How to Write + Run Code

- From PHP script
  → later in today's lecture
- By hand, from command line

RecIng

rec	ing	qty	unit
cake	flour	2.5	cup
cake	milk	3	tbsp
gravy	salt	2	tsp
gravy	sugar	10	g

```
en-cs-cs5142:> sqlite3 test.db
SQLite version 3.6.20
Enter ".help" for instructions
sqlite> CREATE TABLE RecIng(rec VARCHAR(50), ing VARCHAR(50),
  qty FLOAT, unit VARCHAR(10), PRIMARY KEY(rec, ing));
sqlite> INSERT INTO RecIng VALUES('cake','flour',2.5,'cup');
sqlite> INSERT INTO RecIng VALUES('cake','milk',3,'tbsp');
sqlite> INSERT INTO RecIng VALUES('gravy','salt',2,'tsp');
sqlite> SELECT ing, qty FROM RecIng WHERE rec = 'cake';
flour|2.5
milk|3.0
sqlite>
```

sqlite

Create Table Statement

```
CREATE TABLE RecIng(rec VARCHAR(50), ing VARCHAR(50),
  qty FLOAT, unit VARCHAR(10), PRIMARY KEY(rec, ing));
```

createTable ::=
  CREATE [**TEMP**|**TEMPORARY**] **TABLE** (*newTable* | *derivedTable*)
newTable ::=
  [**IF NOT EXISTS**] [*id* **.**] *id* **(** *column*$^+$ (**,** *tableConstraint*)* **)**
derivedTable ::= [*id* **.**] *id* **AS** *select*
column ::= *id* [*type*] ([**CONSTRAINT** *id*] *columnConstraint*)*
columnConstraint ::=
  NOT NULL [*conflict*] | **UNIQUE** [*conflict*]
| **PRIMARY KEY** [*sortOrder*] [*conflict*] [**AUTOINCREMENT**]
| **CHECK** (*expr*) | **DEFAULT** *expr* | **COLLATE** *collationName*
tableConstraint ::=
  PRIMARY KEY (*id*$^+$) [*conflict*]
| **UNIQUE** (*id*$^+$) [*conflict*] | **CHECK** (*expr*)
conflict ::= **ON CONFLICT** *conflictAlgorithm*
conflictAlgorithm ::= **ROLLBACK**|**ABORT**|**FAIL**|**IGNORE**|**REPLACE**

sqlite

Insert Statement

```
INSERT INTO RecIng VALUES('cake','flour',2.5,'cup');
INSERT INTO RecIng VALUES('cake','milk',3,'tbsp');
INSERT INTO RecIng VALUES('gravy','salt',2,'tsp');
```

insert ::=
  INSERT [**OR** *conflictAlgorithm*] **INTO** [*id* **.**] *id* [**(***id****)**] **VALUES(***expr****)**
insert ::=
  INSERT [**OR** *conflictAlgorithm*] **INTO** [*id* **.**] *id* [**(***id****)**] *select*
conflictAlgorithm ::= **ROLLBACK** | **ABORT** | **FAIL** | **IGNORE** | **REPLACE**

sqlite

Select Statement

select ::=
  SELECT [**ALL** | **DISTINCT**] *result*$^+$
  [**FROM** *table* (*joinOp table joinArgs*)*]
  [**WHERE** *expr*]
  [**GROUP BY** *expr*$^+$]
  [**HAVING** *expr*]
  (*compoundOp select*)*
  [**ORDER BY** (*expr* [*sortOrder*])*
  [**LIMIT** *integer* [(**OFFSET** | **,**) *integer*]]
result ::= ***** | *tableName* **.*** | *expr* [[**AS**] *id*]
table ::= *tableName* [**AS** *alias*] | **(***select***)** [**AS** *alias*]
joinOp ::= **,** |
  [**NATURAL**] [**LEFT**|**RIGHT**|**FULL**] [**OUTER**|**INNER**|**CROSS**] **JOIN**
joinArgs ::= [**ON** *expr*] [**USING** **(***id*$^+$**)**]
sortOrder ::= [**COLLATE** *collationName*] [**ASC**|**DESC**]
compoundOp ::= **UNION** | **UNION ALL** | **INTERSECT** | **EXCEPT**

```
SELECT ing, qty
FROM RecIng
WHERE rec = 'cake';
```

List of sqlite Statements

Data Definition Language (DDL)

- (**CREATE** | **ALTER** | **DROP**) **TABLE**
- (**CREATE** | **DROP**) **INDEX**
- (**CREATE** | **DROP**) **TRIGGER**
- (**CREATE** | **DROP**) **VIEW**
- **CREATE VIRTUAL TABLE**
- **ATTACH DATABASE**
- **DETACH DATABASE**
- **ANALYZE**
- **REINDEX**
- **VACUUM**

Data Manipulation Language (DML)

- **INSERT**
- **SELECT**
- **UPDATE**
- **REPLACE**
- **DELETE**
- **BEGIN TRANSACTION**
- **COMMIT TRANSACTION**
- **ROLLBACK TRANSACTION**
- **END TRANSACTION**
- **EXPLAIN**
- **PRAGMA**

Lexical Peculiarities

- Case insensitive
- Commands end with semicolon (`;`)
- Single-line comments: `--`…
- Multi-line comment: `/*`…`*/`
- String literal: `'s'`
  - Escape single quote (`'`) in string with another single quote (`''`), not with backslash (`\'`)
- Identifier: simple *id* or quoted `"`*id*`"`
  - Quoted identifier can contain any character
  - Quoted identifier can be same as keyword

sqlite

Operators

-, **+**, **~**, **NOT**	1		Negation
CASE [*expr*] (**WHEN** *expr* **THEN** *expr*)$^+$ [**ELSE** *expr*] **END**			Conditional
CAST (*expr* **AS** *type***)**	1		Conversion
||	2	L	String concat.
*****, **/**, **%**	2	L	Multiplicative
+, **-**	2	L	Additive
<<, **>>**, **&**, **|**	2	L	Bitwise
<, **<=**, **>**, **>=**	2		Comparison
=, **==**, **!=**, **<>**, **IN**	2		Identity
expr [**NOT**](**LIKE**|**GLOB**|**REGEXP**|**MATCH**) *expr* [**ESCAPE** *expr*]	2/3		Pattern match
expr [**NOT**] **BETWEEN** *expr* **AND** *expr*	3		Comparison
AND, **OR**	2		Logic
[**EXISTS**] (*select*)	1		Query-in-expr

Library Functions

- Aggregate: avg, count, group_concat, max, min, sum, total
- String: glob, length, like, lower, ltrim, quote, replace, rtrim, soundex, substr, trim, upper
- Number: abs, max, min, random, round
- Misc: coalesce, ifnull, hex, last_insert_rowid, load_extension, nullif, randomblob, sqlite_version, typeof, zeroblob
- Date+time: date, time, datetime, julianday, strftime

Reference

SQL Documentation

- Take a class on databases
- Read a standard databases text book
- sqlite: http://www.sqlite.org
- MySQL: http://www.mysql.com
- Tutorial: http://www.w3schools.com/sql/default.asp

Last Slide

- Today's lecture
  - Databases
  - SQL
- Next lecture
  - Session state
  - Form validation
  - AJAX