Relational Model

CSE462 Database Concepts

Demian Lessa

Department of Computer Science and Engineering
State University of New York, Buffalo

January 21–24, 2011

Next...

1 Relational Model

Data Model

- A **data model** is a set of concepts used to describe the structure of the data and the constraints it should obey. It also provides operations for data retrieval and modification.
- In database design, we typically use three different data models.
  - **Conceptual:** concepts are closer to the way users perceive them.
  - **Logical:** falls between the other two, balancing user views with some representation details.
  - **Physical:** representation details such as data organization on disk, available access methods, etc.

Data Model

Why do different data models?

- Requirements Analysis
  - Determines data, applications, critical operations, etc.
- Conceptual Design
  - High-level description of data and constraints (e.g., ERM).
- Logical Design
  - Conversion of the conceptual design into a database schema.
- Schema Refinement
  - Redundancy elimination through a process called normalization.
- Physical Design
  - Considers workloads, indexes, clustering/partitioning, etc.
- Application and Security Design. . .

Overview

Relational Model (1970)

- Originally proposed by Ted Codd.
- Separates physical implementation from conceptual view.
- Models data **independently** from its intended or actual use.
  - Describes data both **minimally** and **mathematically**.
    - A relation describes an association between data items– **tuples** with **attributes**.
  - Uses standard mathematical (logical) operations over the data– **relational algebra** or **relational calculus**.

Definitions

The relational model represents data as two-dimensional **relations**.

title	year	length	genre
Gone With the Wind	1939	231	drama
Star Wars	1977	124	scifi
Wayne's World	1992	95	comedy

Table: The `Movies` relation.

- Each **row** in the `Movies` relation represents a movie and each **column** a movie property. Every column header is an **attribute**.
- A **relation schema** consists of a relation name and a *set of attributes*. Notation: `Movies(title, year, length, genre)`.
- A **database schema** is the set of all relation schemas in the database.

Definitions (cont.)

- Every relation attribute is associated with a **domain**, an elementary type such as `int` or `string`. Domains may optionally be included in relation schemas:

  `Movies(title : string, year : int, length : int, genre : string)`

- Rows of a relation are called **tuples**. A tuple has one component for each relation attribute. Every tuple component must have a value that belongs to the corresponding column's domain or is `NULL` (`NULL` is not a value!). The **arity** of a tuple is the number of components in the tuple. Notation: ('Gone With the Wind', 1939, 231, 'drama').

Definitions (cont.)

- Attributes in a relation schema are a set. A standard order may be specified or else the given order is used.
- Relations are sets of tuples, so presentation order is irrelevant.
- Attributes may be reordered without changing the relation, but tuple components must be reordered accordingly.
- In how many different ways can the `Movies` relation be presented?

year	genre	title	length
1939	drama	Gone With the Wind	231
1977	scifi	Star Wars	124
1992	comedy	Wayne's World	95

Table: Alternate presentation of the `Movies` relation.

Schemas and Instances

- A **relation instance** is the set of tuples of a given relation.
- Instances usually change over time as users insert, delete, and update data. Conventional databases maintain one version of each relation: the current instance. A **database instance** is the set of all relation instances in the database.
- Relation schemas change much less frequently. When modifying a relation schema, all tuples in the relation must be rewritten to accomodate the changes. It may be difficult or impossible to generate appropriate values for new components in existing tuples.
- Users formulate queries against a database schema. Queries are validated against the database schema and evaluated against the database instance.

Key Constraint

- A set of attributes forms a **key** for a relation if no two tuples in a relation instance are allowed to have the same values in all key attributes.
  Notation: `Movies(title, year, length, genre)`.
- A key is **minimal** if the set obtained by removing any attribute from the key is no longer a key.
- A relation may have multiple keys. It may also have no natural key, in which case an artificial (synthetic) one may be created.
- How would you identify:
  - a university student?
  - a company employee?
  - a driver?
  - an automobile?

Required

- Read sections 2.1 and 2.2 of chapter #2.
- Review the movies database schema of section 2.2.8.

Exercise 2.2.1

The relations below constitute part of a banking database.

acctNo	type	balance
12345	savings	12000
23456	checking	1000
34567	savings	25

Table: The `Accounts` relation.

firstName	lastName	idNo	account
Robbie	Banks	901-222	12345
Lena	Hand	805-333	12345
Lena	Hand	805-333	23456

Table: The `Customers` relation.

Specify:

- The attributes of each relation.
- The tuples of each relation.
- The components of the first tuple of each relation.
- The relation schema for each relation.
- The database schema.
- A suitable domain for each attribute.
- Another equivalent way to present each relation.

Exercise 2.2.3:

Considering orders of tuples and attributes, how many different ways are there to represent a relation instance if the instance has:

- Three attributes and three tuples?
- Four attributes and five tuples?
- n attributes and m tuples?

Classwork #1

Web Page	Day	Hits
index.html	2011-01-21	18
schedule.html	2011-01-21	12
syllabus.html	2011-01-21	11
index.html	2011-01-22	18
schedule.html	2011-01-22	9
syllabus.html	2011-01-22	6

Web Statistics: Snapshot of our course's web site statistics.

1. Specify a schema for `WebStats`.
   - Include attribute names, their domains, and a minimal key.
2. Can ("index.html", 2011-01-22, 15) be inserted into `WebStats`?
   - Justify your answer based on your answer above.

SQL

- **Structured Query Language** (**SQL**) is a standardized language used to specify and manipulate relational databases. It consists of a data definition language (DDL) and a data manipulation language (DML). The current standard is SQL:2008.
- SQL defines three kinds of relation– stored relations (tables), computed relations (views), and temporary tables.

SQL: Creating Tables

The `CREATE TABLE` command creates a table by specifying its schema and optional constraints. Simplified syntax:

```
CREATE TABLE tableName (
  attr1 type1 [colum_constraint [...]],
  ...
  attrN typeN [colum_constraint [...]]
  [, table_constraint]
  [, ...]
);
```

Constraints may be specified either as part of an attribute declaration (column constraint) or provided after all attribute declarations (table constraint). Certain constraints must be specified as column constraints (`DEFAULT`, `NOT NULL`) while others as table constraints (multi-column constraints).

SQL: Data Types

SQL data types (not extensive).

- `BIT(n)`, `BIT VARYING(n)`
  - Bit strings of fixed or varying length.
- `BOOLEAN`
  - Logical values, with three truth values: `TRUE`, `FALSE`, `UNKNOWN`.
- `CHAR(n)`, `VARCHAR(n)`
  - Character strings of fixed or varying length.
- `DATE`, `TIME`, `TIMESTAMP`.
  - Temporal values consisting of date, time, or date-and-time:
- Numbers
  - `INT` (also `INTEGER`), `SMALLINT`: integers.
  - `FLOAT` (also `REAL`): single-precision real numbers.
  - `DECIMAL(n,d)`: higher precision real numbers, where `n` is the number of decimal digits and `d` is the number of significant digits to the right of the decimal point.

SQL: Constraints

Specifying `NOT NULL`, `DEFAULT`, and `CHECK` constraints using the `CREATE TABLE` command:

```
CREATE TABLE tableName (
  attr1 type1 [[NOT] NULL] [DEFAULT val1] [CHECK(expr1)],
  ...
  attrN typeN [[NOT] NULL] [DEFAULT valN] [CHECK(exprN)]
  [, [CONSTRAINT chk_name] CHECK(expr)]
  [, ...]
);
```

Constraints:

- `NOT NULL`: tuples must have a value for that attribute at all times.
- `DEFAULT`: the value a tuple component takes if no value is supplied at insertion. If no default value is specified, `NULL` is used.
- `CHECK`: the boolean expression must evaluate to `TRUE` or `UNKNOWN` for all tuples at all times. The expression for a column constraint may only reference that column, but multiple columns for a table constraint.

SQL: Constraints (cont.)

Specifying `PRIMARY KEY` and `UNIQUE` constraints using the `CREATE TABLE` command:

```
CREATE TABLE tableName (
  attr1 type1 [PRIMARY KEY] [UNIQUE],
  ...
  attrN typeN [PRIMARY KEY] [UNIQUE]
  [, [CONSTRAINT pk_name] PRIMARY KEY(attr_list)]
  [, [CONSTRAINT uc_name] UNIQUE(attr_list)]
  [, ...]
);
```

Keys may be declared as `PRIMARY KEY` or `UNIQUE`.

- No two tuples in a relation instance may agree on their key attribute values, unless one of those is `NULL`.
- None of the attributes in a `PRIMARY KEY` may be assigned `NULL`.
- A table may have at most one `PRIMARY KEY` but multiple `UNIQUE` keys.
- Multi-attribute keys must be declared as table constraints.

Example #1

The `Movies` relation, specified with column constraints:

```
CREATE TABLE Movies (
  title  VARCHAR(100) PRIMARY KEY,
  year   INT NOT NULL CHECK(year > 1900),
  length INT CHECK(length > 0),
  genre  VARCHAR(10) DEFAULT 'unknown'
);
```

The `Movies` relation, specified with table constraints:

```
CREATE TABLE Movies (
  title  VARCHAR(100),
  year   INT NOT NULL, -- must be defined here
  length INT,
  genre  VARCHAR(10) DEFAULT 'unknown', -- this one too
  CONSTRAINT pkMovies PRIMARY KEY(title),
  CONSTRAINT chkYearLength CHECK(year > 1900 AND length > 0)
);
```

SQL: Constraints (cont.)

A `FOREIGN KEY` constraint identifies a set of attributes in a **referencing table** that refers to a set of attributes in a **referenced table**. Attributes must be **type compatible** but need not have the same names. Attributes in the referenced table must be a unique or primary key constraint.

Semantics:

- For every tuple t in the referencing table, there exists a unique tuple t' in the referenced table such that the referencing attribute values in t match (except `NULL`) the referenced attribute values in t'.

Observation

- Usually, multiple referencing tuples may refer to the same referenced tuple, reflecting a one-to-many relationship between the tables– the referenced table is the master (“one”) and the referencing table is the child (“many”).

SQL: Constraints (cont.)

Specifying `FOREIGN KEY` constraints using the `CREATE TABLE` command:

```
CREATE TABLE tableName (
  attr1 type1 [REFERENCES refTable (attr_list)],
  ...
  attrN typeN [REFERENCES refTable (attr_list)]
  [, [CONSTRAINT fk_name] FOREIGN KEY(attr_list)
                          REFERENCES refTable (attr_list)]
  [, ...]
);
```

Example #2

The `State` relation: US state names and abbreviations.

```
CREATE TABLE State (
  state  CHAR(2) PRIMARY KEY,
  name   VARCHAR(30) UNIQUE
);
```

The `City` relation: US cities and their associated states and populations.

```
CREATE TABLE City (
  cid         INT PRIMARY KEY,
  name        VARCHAR(100),
  state       CHAR(2),
  population  NUMERIC CHECK(population > 1000),
  CONSTRAINT ucNameState UNIQUE(name,state), -- named
  FOREIGN KEY(state) REFERENCES State(state) -- unnamed
);
```

SQL: Modifying Schemas

- The `DROP TABLE` command removes a table from the database, including all its tuples:

```
DROP TABLE tableName;
```

- The `ALTER TABLE` command allows attributes to be added (1) or dropped (2) from a table:

```
1 ALTER TABLE tableName ADD attr dataType;
2 ALTER TABLE tableName DROP attr; -- by name
```

- The `ALTER TABLE` command also allows constraints to be added (1) or dropped (2) to a table. For example:

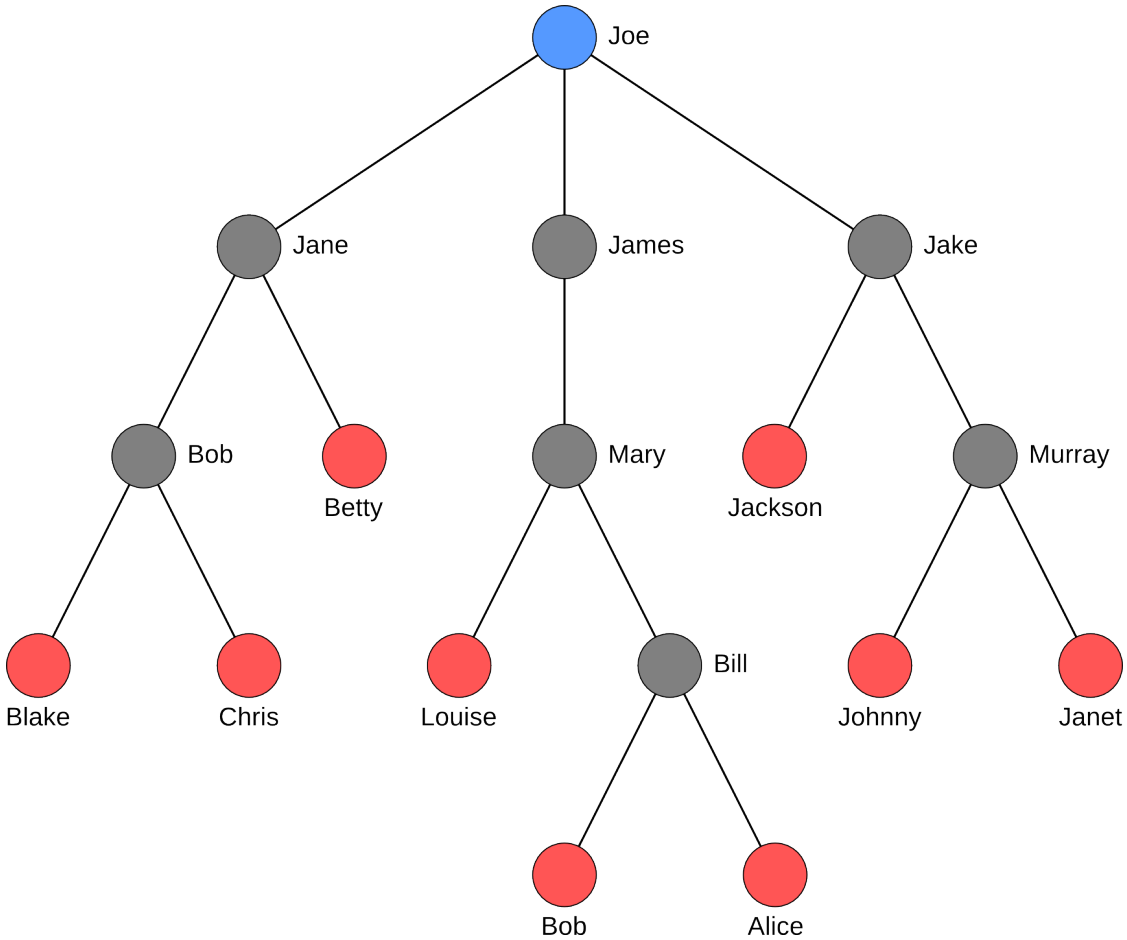
```
1 ALTER TABLE Movies ADD PRIMARY KEY (title, year);
2 ALTER TABLE Movies DROP CONSTRAINT pkMovies; -- by name
```

Required

- Read section 2.3 of chapter #2.
- Answer exercises 2.3.1 and 2.3.2.

Classwork #2

Consider a data model in which all data is modeled as **trees**. Each tree node has **at most one unique parent** and may contain **arbitrary amounts of string data**. Define a relational schema for this model.



Classwork #2 (cont.)

Alternative #1: each node has any number of data values.

- `Node(nid, parentid)`
- `Data(nid, data)`
- Except for `Node.parentid`, all fields are `NOT NULL`.
- `Data.data` is a string, all other fields are integers.

Classwork #2 (cont.)

Alternative #1: each node has any number of data values.

- `Node(nid, parentid)`
- `Data(nid, data)`
- Except for `Node.parentid`, all fields are `NOT NULL`.
- `Data.data` is a string, all other fields are integers.

Alternative #2: each node has any number of key/data pairs.

- `Node(nid, parentid)`
- `Data(nid, key, data)`
- Except for `Node.parentid`, all fields are `NOT NULL`.
- `Data.key` and `Data.data` are strings, all other fields are integers.

Classwork #2 (cont.)

Alternative #1: each node has any number of data values.

- `Node(nid, parentid)`
- `Data(nid, data)`
- Except for `Node.parentid`, all fields are `NOT NULL`.
- `Data.data` is a string, all other fields are integers.

Alternative #2: each node has any number of key/data pairs.

- `Node(nid, parentid)`
- `Data(nid, key, data)`
- Except for `Node.parentid`, all fields are `NOT NULL`.
- `Data.key` and `Data.data` are strings, all other fields are integers.

Alternative #3: ancestry information implicitly encoded.

- `Node(nid, parentid, ibegin, iend)`
- `Data(nid, key, data)`
- All fields are `NOT NULL`.
- `Node.ibegin` and `Node.iend` are reals, `Node.ibegin < Node.iend`.
- `Data.key` and `Data.data` are strings, all other fields are integers.

Classwork #2

- Can you define a simplified relational schema that encodes a relational database?

Classwork #2

- Can you define a simplified relational schema that encodes a relational database?
- **Tip #1:** consider only two relations– one that encodes relation schemas and one that encodes relation instances.

Classwork #2

- Can you define a simplified relational schema that encodes a relational database?
- **Tip #1:** consider only two relations– one that encodes relation schemas and one that encodes relation instances.
- **Tip #2:** assume that schemas consist of relations names with their attributes names and type names.

Classwork #2 (cont.)

Proposed solution:

- `RelSchema(relname, attr_name, attr_type)`
- `RelInstance(relname, attr_name, tid, data)`

Classwork #2 (cont.)

Proposed solution:

- `RelSchema(relname, attr_name, attr_type)`
- `RelInstance(relname, attr_name, tid, data)`

Now answer:

- Why do we need `RelInstance.tid`? `RelInstance.attr_name`?

Classwork #2 (cont.)

Proposed solution:

- `RelSchema(relname, attr_name, attr_type)`
- `RelInstance(relname, attr_name, tid, data)`

Now answer:

- Why do we need `RelInstance.tid`? `RelInstance.attr_name`?
  - `RelInstance.tid` uniquely identifies a tuple within a relation instance.

Classwork #2 (cont.)

Proposed solution:

- `RelSchema(relname, attr_name, attr_type)`
- `RelInstance(relname, attr_name, tid, data)`

Now answer:

- Why do we need `RelInstance.tid`? `RelInstance.attr_name`?
  - `RelInstance.tid` uniquely identifies a tuple within a relation instance.
  - `RelInstance.attr_name` uniquely identifies a component within a tuple.
- Which fields, if any, would you declared as `NOT NULL`?

Classwork #2 (cont.)

Proposed solution:

- `RelSchema(relname, attr_name, attr_type)`
- `RelInstance(relname, attr_name, tid, data)`

Now answer:

- Why do we need `RelInstance.tid`? `RelInstance.attr_name`?
  - `RelInstance.tid` uniquely identifies a tuple within a relation instance.
  - `RelInstance.attr_name` uniquely identifies a component within a tuple.
- Which fields, if any, would you declared as `NOT NULL`?
  - `RelSchema.attr_type`: all attributes must have a type.
  - `RelInstance.data`: no need to assign `NULL`, simply omit the component.
  - All other fields are part of primary keys.
- Would you specify a foreign key for this schema? Explain.

Classwork #2 (cont.)

Proposed solution:

- `RelSchema(relname, attr_name, attr_type)`
- `RelInstance(relname, attr_name, tid, data)`

Now answer:

- Why do we need `RelInstance.tid`? `RelInstance.attr_name`?
  - `RelInstance.tid` uniquely identifies a tuple within a relation instance.
  - `RelInstance.attr_name` uniquely identifies a component within a tuple.
- Which fields, if any, would you declared as `NOT NULL`?
  - `RelSchema.attr_type`: all attributes must have a type.
  - `RelInstance.data`: no need to assign `NULL`, simply omit the component.
  - All other fields are part of primary keys.
- Would you specify a foreign key for this schema? Explain.
  - Yes. `RelInstance` references `RelSchema` on `{relName, attr_name}`.
  - Only data associated with some schema element would be recorded.
- Is it possible to enforce constraints at the database level?

Classwork #2 (cont.)

Proposed solution:

- `RelSchema(relname, attr_name, attr_type)`
- `RelInstance(relname, attr_name, tid, data)`

Now answer:

- Why do we need `RelInstance.tid`? `RelInstance.attr_name`?
  - `RelInstance.tid` uniquely identifies a tuple within a relation instance.
  - `RelInstance.attr_name` uniquely identifies a component within a tuple.
- Which fields, if any, would you declared as `NOT NULL`?
  - `RelSchema.attr_type`: all attributes must have a type.
  - `RelInstance.data`: no need to assign `NULL`, simply omit the component.
  - All other fields are part of primary keys.
- Would you specify a foreign key for this schema? Explain.
  - Yes. `RelInstance` references `RelSchema` on `{relName, attr_name}`.
  - Only data associated with some schema element would be recorded.
- Is it possible to enforce constraints at the database level?
  - In a nutshell: no, it must be enforced on the application side.
  - Longer: advanced features and/or sophisticated encodings may help (in part).

Classwork #2 (cont.)

Feeling adventurous?

- In the Object-Oriented model, data is modeled as objects. Each object belongs to a type (its class), may inherit from one (or more) types, and has a fixed number of typed attributes. Can you come up with a relational schema that encodes an Object-Oriented database?