COURSE OVERVIEW
THE RELATIONAL MODEL

CS121: Introduction to Relational Database Systems

Fall 2014 – Lecture 1

Course Overview

- Introduction to relational database systems
  - Theory and use of relational databases
- Focus on:
  - The Relational Model and relational algebra
  - SQL (the Structured Query Language)
  - The Entity-Relationship model
  - Database schema design and normal forms
  - Various common uses of database systems
- By end of course:
  - Should be very comfortable using relational databases
  - Familiar with basic relational database theory

Textbook

- No textbook is required for the course
  - The lecture slides contain most of the relevant details
  - Other essential materials are provided with the assignments
- A great book: Database System Concepts, 5^{th} ed.
  - Silberschatz, Korth, Sudarshan
  - (The current edition is 6^{th}; they messed a lot of things up…)
  - Covers theory, use, and implementation of relational databases, so good to have for 121/122/123 sequence
- I will also make recordings of the lectures available

Assignments

- Assignments are given approximately weekly
  - Set of problems focusing on that week's material
  - Most include hands-on practice with real databases
  - Made available on Wednesdays
  - Due approx. one week later: Thursdays at 2am
    - That's the start of Thursday, not the end of Thursday
- Midterm and final exam are typically 4-6 hours long
- Assignment and exam weighting:
  - 8 assignments, comprising 65% of your grade
  - Midterm counts for 15% of your grade
  - Final exam counts for 20% of your grade

Course Website and Submissions

- CS121 is on the Caltech Moodle
  - https://courses.caltech.edu/course/view.php?id=1684
  - 2014 enrollment key: unionall (as one word)
- Please enroll in the course as soon as possible!
  - I will make class announcements via Moodle
  - You will submit your assignments via Moodle
- Most assignments will be submitted on the Moodle
  - We suggest you do HW1 and HW5 by hand, rather than on the computer, unless you are awesome at $\LaTeX$
  - (Trust us, you will finish them much faster.)

Grading Policies

- Submit assignments on time!
- Late assignments and exams will be penalized!
  - Up to 1 day (24 hours) late: 10% penalty
  - Up to 2 days (48 hours) late: 30% penalty
  - Up to 3 days (72 hours) late: 60% penalty
  - After 3 days, don't bother. ☹
- But, extensions are available:
  - Must provide a note from Dean's Office or Health Center
  - You also have 3 "late tokens" to use however you want
    - Each late token is worth a 24-hour extension
    - **Can't use late tokens on the final exam without my permission**

Other Administrivia

- I will be away from Caltech for weeks 2 and 3 ☹

- We do have lecture recordings for those weeks
- We will have plenty of TAs to help with the work
  - Will likely have extended office hours for questions during this time

- Will discuss this more next time

Database Terminology

- Database – an organized collection of information
  - A *very* generic term…
  - Covers flat text-files with simple records…
  - …all the way up to multi-TB data warehouses!
  - Some means to query this set of data as a unit, and usually some way to update it as well
- Database Management System (DBMS)
  - Software that manages databases
    - Create, modify, query, backup/restore, etc.
  - Sometimes just "database system"

Before DBMSes Existed…

- Typical approach:
  - Ad-hoc or purpose-built data files
  - Special-built programs implemented various operations against the database
- Want to perform new operations?
  - Create new programs to manipulate the data files!
- Want to change the data model?
  - Update all the programs that access the data!
- How to implement transactions? Security? Integrity constraints?

Enter the DBMS

- Provide layers of abstraction to isolate users, developers from database implementation
  - Physical level: how values are stored/managed on disk
  - Logical level: specification of records and fields
  - View level: queries and operations that users can perform (typically through applications)
- Provide generic database capabilities that specific applications can utilize
  - Specification of database schemas
  - Mechanism for querying and manipulating records

Kinds of Databases

- *Many* kinds of databases, based on usage
- Amount of data being managed
  - embedded databases: small, application-specific systems (e.g. SQLite, BerkeleyDB)
  - data warehousing: vast quantities of data (e.g. Oracle)
- Type/frequency of operations being performed
  - OLTP: Online Transaction Processing
    - "Transaction-oriented" operations like buying a product or booking an airline flight
  - OLAP: Online Analytical Processing
    - Storage and analysis of *very* large amounts of data
    - e.g. "What are my top selling products in each sales region?"

Data Models

- Databases must represent:
  - the data itself (typically structured in some way)
  - associations between different data values
- What kind of data can be modeled?
- What kinds of associations can be represented?
- The data model specifies:
  - what data can be stored (and sometimes how it is stored)
  - associations between different data values
  - what constraints can be enforced
  - how to access and manipulate the data

Data Models (2)

- Most database systems use the <u>relational model</u>
  - A database is a collection of tables containing records
  - Format of records is fixed
    - It can be changed, but this is infrequent!
  - Data is modeled at logical level, not physical level
- Preceded by hierarchical data model, and the network model
  - Very powerful and complicated models
  - Required much more physical-level specification
  - Queries implemented as programs that navigate the schema
  - Schemas couldn't be changed without heavy costs

Data Models

- This course focuses on the Relational Model
  - SQL (Structured Query Language) draws heavily from the relational model
  - Most database systems use the relational model!
- Also focuses on the Entity-Relationship Model
  - Much higher level model than relational model
  - Useful for modeling abstractions
  - Very useful for database design!
  - Not supported by most databases, but used in many database design tools
  - Easy to translate into the relational model

Other Data Models

- Relational model is not the only one in use!
  - By far the most widely used, at this point
- Object model, object-relational model
  - Model data records as "objects" that store references to related objects and values
  - Very similar to the network model, but with a much higher level of abstraction
- XML data models
  - Optimized for XML document storage
  - Queries using XPath, XQuery, etc.
  - XSLT support for transforming XML documents

Other Data Models (2)

- There are also simpler structured storage models
  - Key-value stores, document stores, NoSQL, etc.
  - Relax most of the constraints imposed by relational model
  - Allow for extremely large distributed databases with very flexible schemas
  - (Relational model is one kind of structured storage model)
- Used to manage data for the largest, most heavily used websites
  - Performance and scaling requirements simply disallow the use of the relational model
  - Can't impose constraints without an overwhelming cost

The Relational Model and SQL

Before we start:

- SQL is *loosely* based on the relational model
- Some terms appear in both the relational model and in SQL…

  …but they aren't exactly the same!
- Be careful if you already know some SQL
  - Don't assume that similarly named concepts are identical. They're not!

History of the Relational Model

- Invented by Edgar F. ("Ted") Codd in early 1970s
- Focus was data independence
  - Existing data models required physical level design and implementation
  - Changes were very costly to applications that accessed the database
- IBM, Oracle were first implementers of relational model (1977)
  - Usage spread very rapidly through software industry
  - SQL was a particularly powerful innovation

Relations

- Relations are basically tables of data
  - Each row represents a record in the relation
- A relational database is a set of relations
  - Each relation has a unique name in the database

acct_id	branch_name	balance
A-301	New York	350
A-307	Seattle	275
A-318	Los Angeles	550
…	…	…

The *account* relation

- Each row in the table specifies a <u>relationship</u> between the values in that row
  - The account ID "A-307", branch name "Seattle", and balance "275" are all related to each other

Relations and Attributes

- Each relation has some number of <u>attributes</u>
  - Sometimes called "columns"
- Each attribute has a <u>domain</u>
  - Specifies the set of valid values for the attribute
- The *account* relation:
  - 3 attributes
  - Domain of *balance* is the set of nonnegative integers
  - Domain of *branch_name* is the set of all valid branch names in the bank

acct_id	branch_name	balance
A-301	New York	350
A-307	Seattle	275
A-318	Los Angeles	550
…	…	…

account

Tuples and Attributes

- Each row is called a <u>tuple</u>
  - A fixed-size, ordered set of name-value pairs
- A <u>tuple variable</u> can refer to any valid tuple in a relation
- Each attribute in the tuple has a unique name
- Can also refer to attributes by index
  - Attribute 1 is the first attribute, etc.
- Example:
  - Let tuple variable *t* refer to first tuple in *account* relation
  - *t*[*balance*] = 350
  - *t*[2] = "New York"

acct_id	branch_name	balance
A-301	New York	350
A-307	Seattle	275
A-318	Los Angeles	550
…	…	…

account

Tuples and Relationships

- In the *account* relation:
  - Domain of *acct_id* is D_1
  - Domain of *branch_name* is D_2
  - Domain of *balance* is D_3

acct_id	branch_name	balance
A-301	New York	350
A-307	Seattle	275
A-318	Los Angeles	550
…	…	…

account

- The *account* relation is a subset of the tuples in the Cartesian product $D_1 \times D_2 \times D_3$
- Each tuple included in *account* specifies a <u>relationship</u> between that set of values
  - Hence the name, "relational model"
  - Tuples in the *account* relation specify the details of valid bank accounts

Tuples and Relations

- A relation is a <u>set</u> of tuples
  - Each tuple appears exactly once
    - *Note: SQL tables are multisets! (Sometimes called <u>bags</u>.)*
  - If two tuples t_1 and t_2 have the same values for all attributes, then t_1 and t_2 are the *same tuple* (i.e. $t_1 = t_2$)
- The order of tuples in a relation is not relevant

Relation Schemas

- Every relation has a <u>schema</u>
  - Specifies the type information for relations
  - Multiple relations can have the same schema
- A relation schema includes:
  - an ordered set of attributes
  - the domain of each attribute
- Naming conventions:
  - Relation names are written as all lowercase
  - Relation schema's name is capitalized
- For relation r and relation schema R:
  - Write $r(R)$ to indicate that the schema of r is R

Schema of *account* Relation

- The relation schema of *account* is:

  Account_schema = (*acct_id*, *branch_name*, *balance*)

- To indicate that *account* has schema *Account_schema*:

  account(*Account_schema*)

acct_id	branch_name	balance
A-301	New York	350
A-307	Seattle	275
A-318	Los Angeles	550
…	…	…

account

- Important note:
  - Domains are not stated explicitly in this notation!

Relation Schemas

- Relation schemas are ordered <u>sets</u> of attributes
  - Can use set operations on them
- Examples:

  Relations $r(R)$ and $s(S)$
    - Relation r has schema R
    - Relation s has schema S

  $R \cap S$
    - The set of attributes that R and S have in common

  $R - S$
    - The set of attributes in R that are not also in S

  $K \subseteq R$
    - K is some subset of the attributes in relation schema R

Attribute Domains

- The relational model constrains attribute domains to be <u>atomic</u>
  - Values are indivisible units
- Mainly a simplification
  - Virtually all relational database systems provide non-atomic data types
- Attribute domains may also include the <u>*null*</u> value
  - *null* = the value is unknown or unspecified
  - *null* can often complicate things. Generally considered good practice to avoid wherever reasonable to do so.

Relations and Relation Variables

- More formally:
- *account* is a relation variable
  - A name associated with a specific schema, and a set of tuples that satisfies that schema
  - (sometimes abbreviated "relvar")
- A specific set of tuples with the same schema is called a relation value (sometimes abbreviated "relval")
  - (Formally, this can also be called a relation)
  - Can be associated with a relation variable
  - Or, can be generated by applying relational operations to one or more relation variables

acct_id	branch_name	balance
A-301	New York	350
A-307	Seattle	275
A-318	Los Angeles	550
…	…	…

The *account* relation

Relations and Relation Variables (2)

- Problem:
  - The term "relation" is often used in slightly different ways
- "Relation" normally means the collection of tuples
  - i.e. "relation" usually means "relation value"
- It is often used less formally to refer to a relation variable and its associated relation value
  - e.g. "the *account* relation" is really a relation variable that holds a specific relation value

acct_id	branch_name	balance
A-301	New York	350
A-307	Seattle	275
A-318	Los Angeles	550
…	…	…

The *account* relation

Distinguishing Tuples

- Relations are *sets* of tuples…
  - No two tuples can have the same values for *all* attributes…
  - But, some tuples might have the same values for *some* attributes
- Example:
  - Some accounts have the same balance
  - Some accounts are at the same branch

acct_id	branch_name	balance
A-301	New York	350
A-307	Seattle	275
A-318	Los Angeles	550
A-319	New York	80
A-322	Los Angeles	275

account

Keys

- Keys are used to distinguish individual tuples
  - A superkey is a set of attributes that uniquely identifies tuples in a relation

acct_id	branch_name	balance
A-301	New York	350
A-307	Seattle	275
A-318	Los Angeles	550
A-319	New York	80
A-322	Los Angeles	275

account

- Example:
  {*acct_id*} is a superkey
- Is {*acct_id*, *balance*} a superkey?
  - Yes! Every tuple will have a unique set of values for this combination of attributes.
- Is {*branch_name*} a superkey?
  - No. Each branch can have multiple accounts

Superkeys and Candidate Keys

- A superkey is a set of attributes that uniquely identifies tuples in a relation
- Adding attributes to a superkey produces another superkey
  - If {*acct_id*} is a superkey, so is {*acct_id*, *balance*}
  - If a set of attributes $K \subseteq R$ is a superkey, so is any superset of K
  - Not all superkeys are equally useful…
- A *minimal* superkey is called a <u>candidate key</u>
  - A superkey for which no proper subset is a superkey
  - For *account*, only {*acct_id*} is a candidate key

Primary Keys

- A relation might have several candidate keys
- In these cases, one candidate key is chosen as the primary means of uniquely identifying tuples
  - Called a primary key
- Example: *customer* relation
  - Candidate keys could be:

    {*cust_id*}

    {*cust_ssn*}
  - Choose primary key:

    {*cust_id*}

cust_id	cust_name	cust_ssn
23-652	Joe Smith	330-25-8822
15-202	Ellen Jones	221-30-6551
23-521	Dave Johnson	005-81-2568
…	…	…

customer

Primary Keys (2)

- Keys are a property of the relation schema, not individual tuples
  - Applies to *all* tuples in the relation
- Primary key attributes are listed first in relation schema, and are underlined
- Examples:

  Account_schema = (*<u>acct_id</u>*, *branch_name*, *balance*)

  Customer_schema = (*<u>cust_id</u>*, *cust_name*, *cust_ssn*)
- Only indicate primary keys in this notation
  - Other candidate keys are not specified

Primary Keys (3)

- Multiple records cannot have the same values for a primary key!
  - …or any candidate key, for that matter…
- Example: *customer*(*cust_id*, *cust_name*, *cust_ssn*)

	cust_id	cust_name	cust_ssn
	23-652	Joe Smith	330-25-8822
	15-202	Ellen Jones	221-30-6551
	23-521	Dave Johnson	005-81-2568
✕	15-202	Albert Stevens	450-22-5869
	…	…	…

customer

  - Two customers cannot have the same ID.
- This is an example of an invalid relation
  - Set of tuples doesn't satisfy the required constraints

Keys Constrain Relations

- Primary keys *constrain* the set of tuples that can appear in a relation
  - Same is true for *all* superkeys
- For a relation r with schema R
  - If $K \subseteq R$ is a superkey then
    $\langle \forall\ t_1, t_2 \in r(R) : t_1[K] = t_2[K] : t_1[R] = t_2[R] \rangle$
  - i.e. if two tuple-variables have the same values for the superkey attributes, then they refer to the same tuple
    - $t_1[R] = t_2[R]$ is equivalent to saying $t_1 = t_2$

Choosing Candidate Keys

- Since candidate keys constrain the tuples that can be stored in a relation…
  - Attributes that would make good (or bad) candidate keys depend on *what is being modeled*
- Example: customer name as candidate key?
  - Very likely that multiple people will have same name
  - Thus, not a good idea to use it as a candidate key
- These constraints motivated by external requirements
  - Need to understand what we are modeling in the database

Foreign Keys

- One relation schema can include the attributes of another schema's primary key
- Example: *depositor* relation
  - *Depositor_schema* = (*cust_id*, *acct_id*)
  - Associates customers with bank accounts
  - *cust_id* and *acct_id* are both <u>foreign keys</u>
    - *cust_id* references the primary key of *customer*
    - *acct_id* references the primary key of *account*
  - *depositor* is the <u>referencing relation</u>
    - It refers to the *customer* and *account* relations
  - *customer* and *account* are the <u>referenced relations</u>

depositor Relation

cust_id	cust_name	cust_ssn
23-652	Joe Smith	330-25-8822
15-202	Ellen Jones	221-30-6551
23-521	Dave Johnson	005-81-2568
…	…	…

customer

acct_id	branch_name	balance
A-301	New York	350
A-307	Seattle	275
A-318	Los Angeles	550
…	…	…

account

cust_id	acct_id
15-202	A-301
23-521	A-307
23-652	A-318
…	…

depositor

- *depositor* relation references *customer* and *account*
- Represents relationships between customers and their accounts
- Example: Joe Smith's accounts
  - "Joe Smith" has an account at the "Los Angeles" branch, with a balance of 550.

Foreign Key Constraints

- Tuples in *depositor* relation specify values for *cust_id*
  - *customer* relation <u>must</u> contain a tuple corresponding to each *cust_id* value in *depositor*
- Same is true for *acct_id* values and *account* relation
- Valid tuples in a relation are also constrained by foreign key references
  - Called a <u>foreign-key constraint</u>
- Consistency between two dependent relations is called <u>referential integrity</u>
  - Every foreign key value must have a corresponding primary key value

Foreign Key Constraints (2)

- Given a relation $r(R)$
  - A set of attributes $K \subseteq R$ is the primary key for R
- Another relation $s(S)$ references r
  - $K \subseteq S$ too
  - $\langle\ \forall\ t_s \in s : \exists\ t_r \in r : t_s[K] = t_r[K]\ \rangle$
- Notes:
  - K is not required to be a candidate key for S, only R
  - K may also be part of a larger candidate key for S

Primary Key of *depositor* Relation?

- *Depositor_schema* = (*cust_id*, *acct_id*)
- If {*cust_id*} is the primary key:
  - A customer can only have one account
    - Each customer's ID can appear only once in *depositor*
  - An account could be owned by multiple customers
- If {*acct_id*} is the primary key:
  - Each account can be owned by only one customer
    - Each account ID can appear only once in *depositor*
  - Customers could own multiple accounts
- If {*cust_id*, *acct_id*} is the primary key:
  - Customers can own multiple accounts
  - Accounts can be owned by multiple customers
- Last option is how most banks really work

cust_id	acct_id
15-202	A-301
23-521	A-307
23-652	A-318
…	…

depositor