Course Overview

• Introduction to relational database systems
• Focus on:
  – The theory behind relational database systems
  – The Relational Model
  – 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 comfortable using relational databases
  – Should be familiar with basic relational database theory
Book, Assignments

• Book: *Database System Concepts, 5th ed.*
  – Silberschatz, Korth, Sudarshan
  – A copy should be on reserve in SFL
• Assignments are given approximately weekly
  – Reading from Database System Concepts, etc.
  – Set of problems focusing on that week’s material
  – Due approx. one week later
  – Late penalty: 25% off per day late!
• Midterm and final exam
Long-Term Overview

• Can take this term by itself to learn theory/usage
• This is first term in a two term course
  – Second term focuses on implementation of relational database systems (the really fun stuff!)
• Necessary prerequisite is to understand the theory and the relational model
  – All modern relational databases are based on this material
  – After second term of course, should have no problem working on relational database implementations
    • Oracle, Sybase, MS SQLServer, PostgreSQL, …
Database Terminology

• Database – a collection of information
  – A very generic term…
  – Covers flat text-files of records…
  – All the way up to multi-TB data-warehouses

• Database Management System (DBMS)
  – Software that manages databases
    • Create, modify, query, backup/restore, etc.
  – Sometimes just “database system”
Kinds of Databases

- *Many* kinds of databases, based on usage

- Amount of data being managed
  - embedded databases: small, app-specific systems
  - data-warehousing: vast quantities of data

- Type and frequency of operations being performed
  - OLAP: Online Analytical Processing
    - Managing customer or product information in a business
    - Logging and analysis of user data
  - OLTP: Online Transaction Processing
Kinds of Databases (2)

• What data model does the database use?
  – What kinds of data can the database represent?
  – What kind of associations between different items?
• Most common model (currently) is the relational model
  – Most database applications use relational databases
• Other models too:
  – Older models: hierarchical model, network model
  – Newer models: object or object-relational databases
  – Also, XML databases
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 and managed
  – 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
The Data Model

• 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?
• Data model specifies:
  – what data can be stored (and sometimes how)
  – associations between different data values
  – what constraints can be enforced
  – how to access and manipulate the data
The Data Model (2)

- Most database systems use the **relational model**
  - Record-based model
  - 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 complex models
  - Required much more physical-level specification
  - 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 database design tools
  - Easy to translate into the relational model
The Relational Model
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. 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 collection of relations
  – Each relation has a unique name

• Each row in the table specifies a **relationship** between the values in that row
  – The account ID “A-307”, branch name “Seattle”, and balance “275” are all related to each other

<table>
<thead>
<tr>
<th>acct_id</th>
<th>branch_name</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-301</td>
<td>New York</td>
<td>350</td>
</tr>
<tr>
<td>A-307</td>
<td>Seattle</td>
<td>275</td>
</tr>
<tr>
<td>A-318</td>
<td>Los Angeles</td>
<td>550</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Relations and Attributes

• Each relation has some number of attributes
  – Sometimes called “columns”
• Each attribute has a domain
  – 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 branch names in the bank

<table>
<thead>
<tr>
<th>acct_id</th>
<th>branch_name</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-301</td>
<td>New York</td>
<td>350</td>
</tr>
<tr>
<td>A-307</td>
<td>Seattle</td>
<td>275</td>
</tr>
<tr>
<td>A-318</td>
<td>Los Angeles</td>
<td>550</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Tuples and Attributes

• Each row is called a **tuple**
  – A fixed-size, ordered list of values
• A **tuple variable** can refer to any valid tuple in a relation
• Each attribute in the tuple has a unique name
• Can also refer to attributes by number
  – Attribute 1 is the first attribute, etc.
• Example:
  – Tuple variable $t$ refers to first tuple in `account` relation
  – $t[balance] = 350$
  – $t[2] = \text{“New York”}$

<table>
<thead>
<tr>
<th>acct_id</th>
<th>branch_name</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-301</td>
<td>New York</td>
<td>350</td>
</tr>
<tr>
<td>A-307</td>
<td>Seattle</td>
<td>275</td>
</tr>
<tr>
<td>A-318</td>
<td>Los Angeles</td>
<td>550</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
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 $D_3$
- 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 relationship between that set of values
  - Hence the name, “relational model”
  - Tuples in account relation specify the details of valid bank accounts

<table>
<thead>
<tr>
<th>acct_id</th>
<th>branch_name</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-301</td>
<td>New York</td>
<td>350</td>
</tr>
<tr>
<td>A-307</td>
<td>Seattle</td>
<td>275</td>
</tr>
<tr>
<td>A-318</td>
<td>Los Angeles</td>
<td>550</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Tuples and Relations

• A relation is a set of tuples
  – Each tuple appears exactly once
    • Note: SQL tables are multisets!
  – 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 schema
  – Specifies the type information for relations
  – Multiple relations can have the same schema

• A relation schema includes:
  – a set of attributes
  – the domain of each attribute

• Naming:
  – Relation names written as all lowercase
  – Relation schema’s name is capitalized

• For relation $r$ and relation schema $R$:
  – Write $r(R)$ to indicate that schema of $r$ is $R$
Schema of *account* Relation

- The relation schema of *account* is:
  \[ \text{Account\_schema} = (\text{acct\_id}, \text{branch\_name}, \text{balance}) \]

- To indicate that *account* has schema *Account\_schema*:
  \[ \text{account}(\text{Account\_schema}) \]

- Note:
  - Domains aren’t typically stated explicitly in this notation…

<table>
<thead>
<tr>
<th>acct_id</th>
<th>branch_name</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-301</td>
<td>New York</td>
<td>350</td>
</tr>
<tr>
<td>A-307</td>
<td>Seattle</td>
<td>275</td>
</tr>
<tr>
<td>A-318</td>
<td>Los Angeles</td>
<td>550</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Relation Schemas

• Relation schemas are sets 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 **atomic**
  – Values are indivisible units

• Mainly a simplification
  – Virtually all relational database systems provide non-atomic data types

• **All** attribute domains include the **null** value
  – *null* = the value is unknown or unspecified
  – Even though all domains contain *null* value, relations might not contain any tuples with *null* values…
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

<table>
<thead>
<tr>
<th>acct_id</th>
<th>branch_name</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-301</td>
<td>New York</td>
<td>350</td>
</tr>
<tr>
<td>A-307</td>
<td>Seattle</td>
<td>275</td>
</tr>
<tr>
<td>A-318</td>
<td>Los Angeles</td>
<td>550</td>
</tr>
<tr>
<td>A-319</td>
<td>New York</td>
<td>80</td>
</tr>
<tr>
<td>A-322</td>
<td>Los Angeles</td>
<td>275</td>
</tr>
</tbody>
</table>
Keys

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

- Example:
  - `{ acct_id }` is a superkey

- Is `{ acct_id, balance }` a superkey?
  - Yes! Every tuple has 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 candidate key
  – 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_name, address }
  – Choose primary key:
    { cust_id }

<table>
<thead>
<tr>
<th>cust_id</th>
<th>cust_name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>23-521</td>
<td>Joe Smith</td>
<td>123 Evergreen Way</td>
</tr>
<tr>
<td>15-202</td>
<td>Ellen Jones</td>
<td>35 Market St. #214</td>
</tr>
<tr>
<td>23-521</td>
<td>Dave Johnson</td>
<td>621 Washington Rd.</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

"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
  – Key attributes are also underlined
• Examples:
  \[ \text{Account\_schema} = (\text{acct\_id}, \text{branch\_name}, \text{balance}) \]
  \[ \text{Customer\_schema} = (\text{cust\_id}, \text{cust\_name}, \text{address}) \]
• Only indicate primary keys in this notation
  – Other candidate keys are not specified
Primary Keys (3)

• Multiple records can’t have the same values for a primary key!

• Example:

<table>
<thead>
<tr>
<th>cust_id</th>
<th>cust_name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>23-652</td>
<td>Joe Smith</td>
<td>123 Evergreen Way</td>
</tr>
<tr>
<td>15-202</td>
<td>Ellen Jones</td>
<td>35 Market St. #214</td>
</tr>
<tr>
<td>23-521</td>
<td>Dave Johnson</td>
<td>621 Washington Rd.</td>
</tr>
<tr>
<td>15-202</td>
<td>Albert Stevens</td>
<td>92 Terrace Lane</td>
</tr>
</tbody>
</table>

– Two customers cannot have the same ID.
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
    \[
    \langle \forall \ t_1, t_2 \in R : t_1[K] = t_2[K] : t_1[R] = t_2[R] \rangle
    \]
  – i.e. if two tuples have the same values for the superkey attrs, then they are the same tuple
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 are motivated by external requirements
  – Need to understand what is being modeled
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 foreign keys
    • cust_id references the primary key of customer
    • acct_id references the primary key of account
  – depositor is the referencing relation
    • It refers to the customer and account relations
  – customer and account are the referenced relations
**depositor Relation**

<table>
<thead>
<tr>
<th>cust_id</th>
<th>cust_name</th>
<th>address</th>
</tr>
</thead>
<tbody>
<tr>
<td>23-652</td>
<td>Joe Smith</td>
<td>123 Evergreen Way</td>
</tr>
<tr>
<td>15-202</td>
<td>Ellen Jones</td>
<td>35 Market St. #214</td>
</tr>
<tr>
<td>23-521</td>
<td>Dave Johnson</td>
<td>621 Washington Rd.</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>acct_id</th>
<th>branch_name</th>
<th>balance</th>
</tr>
</thead>
<tbody>
<tr>
<td>A-301</td>
<td>New York</td>
<td>350</td>
</tr>
<tr>
<td>A-307</td>
<td>Seattle</td>
<td>275</td>
</tr>
<tr>
<td>A-318</td>
<td>Los Angeles</td>
<td>550</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

- **depositor relation references**
  - *customer* and *account*

- Represents relationships between customers and accounts
  - “Joe Smith” has an account at the “Los Angeles” branch, with a balance of 550.

<table>
<thead>
<tr>
<th>cust_id</th>
<th>acct_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>15-202</td>
<td>A-301</td>
</tr>
<tr>
<td>23-521</td>
<td>A-307</td>
</tr>
<tr>
<td>23-652</td>
<td>A-318</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
Foreign Key Constraints

- Tuples in `depositor` relation specify values for `cust_id`
  - `customer` relation must 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 foreign-key constraint
- Consistency between two dependent relations is called referential integrity
  - Every foreign key value has a corresponding primary key value
Foreign Key Constraints (2)

• Given a relation $r(R)$
  – $K \subseteq R$ is the primary key for $R$
• Another relation $s(S)$ references $r$
  – $K \subseteq S$ too
  – $\langle \forall t_s \in s : \exists t_r \in r : t_s[K] = t_r[K] \rangle$
Primary Key of \textit{depositor} Relation?

- $\text{Depositor\_schema} = (\text{cust\_id}, \text{acct\_id})$
- If $\{ \text{cust\_id} \}$ is the primary key:
  - A customer can only have one account
    - Each customer’s ID can appear only once in $\text{depositor}$
  - An account could be owned by multiple customers
- If $\{ \text{acct\_id} \}$ is the primary key:
  - Each account can be owned by only one customer
    - Each account ID can appear only once in $\text{depositor}$
    - Customers could own multiple accounts
- If $\{ \text{cust\_id}, \text{acct\_id} \}$ is the primary key:
  - Customers can own multiple accounts
  - Accounts can be owned by multiple customers
- Last option is how most banks really work
Review

• Database Management Systems (DBMSes) provide generic capabilities for creating and managing databases
• Most modern database systems follow the relational model
• Relations are sets of tuples that specify relationships between values
• Each relation has a schema specifying the attributes, and each attribute’s domain of valid values
• Candidate keys and foreign keys further constrain the valid tuples in a relation
Next Time

• The Relational Algebra
  – How to retrieve information from a set of relations
  – How to modify the contents of a relation