Goals

- Understand:
  - The relational model
  - Relational model terminology
- Write SQL statements to create tables
Why Study the Relational Model?

- Most widely used model.
  - Vendors: IBM, Microsoft, Oracle, Sybase, etc.
- Competitors:
  - Object-Oriented model
    - ObjectStore, Versant, Ontos
  - A synthesis: object-relational model
    - Informix Universal Server, Oracle, DB2
  - XML
  - Key-value stores

Relational Database

- A relation is a two-dimensional table
- Relation schema describes the structure for the table
  - Relation name
  - Column names
  - Column types
- A relational database is a set of relations
Relation Example

EMPLOYEE(EmployeeNumber:integer,
    FirstName:string,
    LastName:string,
    Department:string,
    Email:string,
    Phone:integer)

<table>
<thead>
<tr>
<th>EmployeeNumber</th>
<th>FirstName</th>
<th>LastName</th>
<th>Department</th>
<th>Email</th>
<th>Phone</th>
</tr>
</thead>
<tbody>
<tr>
<td>100</td>
<td>Jerry</td>
<td>Johnson</td>
<td>Accounting</td>
<td><a href="mailto:JJ@somewhere.com">JJ@somewhere.com</a></td>
<td>236-9987</td>
</tr>
<tr>
<td>200</td>
<td>Mary</td>
<td>Abemathy</td>
<td>Finance</td>
<td><a href="mailto:MA@somewhere.com">MA@somewhere.com</a></td>
<td>444-8898</td>
</tr>
<tr>
<td>300</td>
<td>Liz</td>
<td>Smathers</td>
<td>Finance</td>
<td><a href="mailto:LS@somewhere.com">LS@somewhere.com</a></td>
<td>777-0098</td>
</tr>
<tr>
<td>400</td>
<td>Tom</td>
<td>Caruthers</td>
<td>Accounting</td>
<td><a href="mailto:TC@somewhere.com">TC@somewhere.com</a></td>
<td>236-9987</td>
</tr>
<tr>
<td>500</td>
<td>Tom</td>
<td>Jackson</td>
<td>Production</td>
<td><a href="mailto:TJ@somewhere.com">TJ@somewhere.com</a></td>
<td>444-9980</td>
</tr>
<tr>
<td>600</td>
<td>Eleanor</td>
<td>Caldera</td>
<td>Legal</td>
<td><a href="mailto:EC@somewhere.com">EC@somewhere.com</a></td>
<td>767-0900</td>
</tr>
<tr>
<td>700</td>
<td>Richard</td>
<td>Bandalone</td>
<td>Legal</td>
<td><a href="mailto:RB@somewhere.com">RB@somewhere.com</a></td>
<td>767-0900</td>
</tr>
</tbody>
</table>

Relation

- All entries in a column are of the same kind
- Each column has a unique name
- Cells of the table hold a single value
- The order of the columns is not important
- The order of the rows is not important
- No two rows may be identical
Tables That Are Not Relations

Although not all tables are relations, the terms table and relation are normally used interchangeably.

The following sets of terms are used interchangeably:

<table>
<thead>
<tr>
<th>Table</th>
<th>Column</th>
<th>Row</th>
</tr>
</thead>
<tbody>
<tr>
<td>Relation</td>
<td>Attribute</td>
<td>Tuple</td>
</tr>
<tr>
<td>File</td>
<td>Field</td>
<td>Record</td>
</tr>
</tbody>
</table>
Create a Table

- **CREATE TABLE** statement is used for creating relations/tables
- Each column is described with three parts:
  - column name
  - data type
  - optional constraints

Specify Data Types

- Choose the most specific data type possible!!!

- Generic Data Types:
  - CHAR(n)
  - VARCHAR(n)
  - DATE
  - TIME
  - MONEY
  - INTEGER
  - DECIMAL

```sql
CREATE TABLE EMPLOYEE (  
    EmployeeNumber integer,  
    EmployeeName char(50),  
    Phone char(15),  
    Email char(50),  
    HireDate date,  
    ReviewDate date  
)  
```
Specify Null Status

- **Null status**: whether or not the value of the column can be NULL

```sql
CREATE TABLE EMPLOYEE (
    EmployeeNumber integer NOT NULL,
    EmployeeName char (50) NOT NULL,
    Phone char (15) NULL,
    Email char(50) NULL,
    HireDate date NOT NULL,
    ReviewDate date NULL
)
```

Specify Default Values

- **Default value** - value supplied by the DBMS, if no value is specified when a row is inserted

```sql
CREATE TABLE EMPLOYEE (
    EmployeeNumber integer NOT NULL,
    EmployeeName char (50) NOT NULL,
    Phone char (15) NULL,
    Email char(50) NULL,
    HireDate date NOT NULL DEFAULT (getdate()),
    ReviewDate date NULL
)
```
Specify Other Data Constraints

- **Data constraints** are limitations on data values

```sql
CREATE TABLE EMPLOYEE(
    EmployeeNumber integer NOT NULL,
    EmployeeName char (50) NOT NULL,
    Phone char (15) NULL,
    Email char(50) NULL,
    HireDate date NOT NULL DEFAULT (getdate()),
    ReviewDate date NULL,
    CONSTRAINT Check_Email CHECK (Email LIKE '%@gmail.com')
)
```

Integrity Constraints (IC)

- IC: condition that must be true for any instance of the database
  - Domain constraints
  - Key constraints
  - Foreign Key constraints
- ICs are **specified** when schema is defined
- ICs are **checked** when relations are modified
- A **legal instance** of a relation is one that satisfies all specified ICs
  - DBMS should not allow illegal instances
Keys

- A **key** is a combination of one or more columns that is used to identify rows in a relation.
- A **composite key** is a key that consists of two or more columns.
- A set of columns is a **key** for a relation if:
  1. No two distinct rows can have same values in all key columns, and
  2. This is not true for any subset of the key.
- Part 2 false? A **superkey**

Candidate and Primary Keys

- A **candidate key** is a key.
- A **primary key** is a candidate key selected as the primary means of identifying rows in a relation:
  - There is one and only one primary key per relation.
  - The primary key may be a composite key.
  - The ideal primary key is short, numeric and never changes.
Surrogate Keys

- A **surrogate key** is an artificial column added to a relation to serve as a primary key:
  - DBMS supplied
  - Short, numeric and never changes – an ideal primary key!
  - Has artificial values that are meaningless to users
- Remember Access (ID – auto number)

Specify Primary Key

```sql
CREATE TABLE EMPLOYEE (  
    EmployeeNumber integer NOT NULL,  
    EmployeeName char (50) NOT NULL,  
    Phone char (15) NULL,  
    Email char(50) NULL,  
    HireDate date NOT NULL DEFAULT (getdate()),  
    ReviewDate date NULL,  
    CONSTRAINT Check_Email CHECK (Email LIKE '%@gmail.com'),  
    CONSTRAINT PK_Employee PRIMARY KEY (EmployeeNumber)
)  
```
Specify Alternate Keys

- **Alternate keys**: alternate identifiers of unique rows in a table

```sql
CREATE TABLE EMPLOYEE (  
    EmployeeNumber integer NOT NULL,
    EmployeeName char (50) NOT NULL,
    Phone char (15) NULL,
    Email char(50) NULL,
    HireDate date NOT NULL DEFAULT (getdate()),
    ReviewDate date NULL,
    CONSTRAINT Check_Email CHECK (Email LIKE '%@gmail.com'),
    CONSTRAINT PK_Employee PRIMARY KEY (EmployeeNumber),
    CONSTRAINT AK_Email UNIQUE (Email),
    CONSTRAINT AK_ENamePhone UNIQUE (EmployeeName, Phone)
)
```

**ICE: Is This a Relation? Why?**

<table>
<thead>
<tr>
<th>A</th>
<th>X</th>
<th>C</th>
<th>A</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Ryan</td>
<td>MD</td>
<td><a href="mailto:jr@gmail.com">jr@gmail.com</a></td>
</tr>
<tr>
<td>Bob</td>
<td>Smith</td>
<td>MD, VA, NY</td>
<td><a href="mailto:bsm@gmail.com">bsm@gmail.com</a></td>
</tr>
<tr>
<td>Alice</td>
<td>Brown</td>
<td>CA</td>
<td></td>
</tr>
<tr>
<td>Jane</td>
<td>Doe</td>
<td>WA</td>
<td><a href="mailto:jd@yahoo.com">jd@yahoo.com</a></td>
</tr>
<tr>
<td>John</td>
<td>Ryan</td>
<td>MD</td>
<td><a href="mailto:jr@gmail.com">jr@gmail.com</a></td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>5</td>
<td>4</td>
</tr>
</tbody>
</table>
ICE: Find possible PK, AK

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>X</td>
<td>Y</td>
<td>Z</td>
<td>W</td>
</tr>
<tr>
<td>John</td>
<td>Ryan</td>
<td>MD</td>
<td><a href="mailto:jr@gmail.com">jr@gmail.com</a></td>
</tr>
<tr>
<td>Bob</td>
<td>Smith</td>
<td>MD</td>
<td><a href="mailto:bsm@gmail.com">bsm@gmail.com</a></td>
</tr>
<tr>
<td>Alice</td>
<td>Brown</td>
<td>CA</td>
<td></td>
</tr>
<tr>
<td>John</td>
<td>Doe</td>
<td>WA</td>
<td><a href="mailto:jd@yahoo.com">jd@yahoo.com</a></td>
</tr>
</tbody>
</table>

Foreign Keys and Referential Integrity Constraints

- A **foreign key** is the primary key of one relation that is placed in another relation to form a link between the relations
- A **referential integrity constraint**: the values of the foreign key must exist as primary key values in the corresponding relation → No ‘dangling references’
Foreign Keys

CREATE TABLE LOCKER(
    LockerNumber integer PRIMARY KEY,
    LockerRoom integer,
    LockerSize integer)

CREATE TABLE CLUB_MEMBER(
    MemberNumber integer PRIMARY KEY,
    MemberName char(50),
    Phone char(15),
    Email char(50),
    LockerNumber integer NULL,
    CONSTRAINT FK_Locker FOREIGN KEY (LockerNumber) REFERENCES LOCKER(LockerNumber),
    CONSTRAINT Unique_Locker UNIQUE(LockerNumber))

Enforcing Referential Integrity

- What if a new “Member” row is added/updated that references a non-existent locker?
  - Reject it!
- What if a Locker row is deleted? Options:
  - Also delete all Member rows that refer to it.
  - Disallow deletion of Locker row that is referred.
  - Set `LockerNumber in Member` to default value
  - Set `LockerNumber in Member` to `null`
- Similar if primary key of Locker row is updated
Referential Integrity in SQL/92

- SQL/92 supports all 4 options on deletes and updates.
  - Default is NO ACTION (delete/update is rejected)
  - CASCADE (delete/update all rows that refer to deleted/updated row)
  - SET NULL / SET DEFAULT

CREATE TABLE CLUB_MEMBER(
    MemberNumber integer PRIMARY KEY
    MemberName char(50),
    Phone char(15),
    Email char(50),
    LockerNumber integer NULL,
    CONSTRAINT FK_Locker FOREIGN KEY (LockerNumber) REFERENCES
    LOCKER(LockerNumber) ON DELETE SET NULL ON UPDATE
    CASCADE,
    CONSTRAINT Unique_Locker UNIQUE(LockerNumber)
)
Summary – Relational Model

- 2-D tables
- Relational schema: structure of table
- Constraints
  - Domain
  - Key
    - Candidate, Primary, Alternate, Surrogate
    - Foreign key – Referential integrity constraint

SQL: Creating Tables

CREATE TABLE table_name(
  column_name1 column_type1 [constraints1],
  ...
  [CONSTRAINT constraint_name] table_constraint
)

Table constraints:
- NULL/NOT NULL
- PRIMARY KEY (columns)
- UNIQUE (columns)
- CHECK (conditions)
- FOREIGN KEY (local_columns) REFERENCES foreign_table (foreign_columns) [ON DELETE action_d ON UPDATE action_u]

Specify surrogate key in SQL Server:
  column_name int_type IDENTITY (seed, increment)
Specify surrogate key in MySQL:
  column_name int_type AUTO_INCREMENT