Lecture 2:
- Relational Model (Part I: Representing data)

Homework:
- Quiz 1 Tues
Fictional e-scooter/e-bike sharing company ("micromobility")

- loosely based on parts of Global Bikeshare Feed Specification (GBFS)
- which is used by Lime(*) and Bird
- we'll use the running example in class and in some homeworks

Basic data stored:

- Vehicle information (types of vehicles)
- "Real-time" vehicle usage data
- Customer information
- Price plans
- Customer trip information

(*) e.g.: data.lime.bike/api/partners/v1/gbfs/seattle/free_bike_status.json
Relational Model: Relations

VehicleType

<table>
<thead>
<tr>
<th>vt_id</th>
<th>make</th>
<th>model</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Segway</td>
<td>Ninebot</td>
</tr>
<tr>
<td>2</td>
<td>Lime-S</td>
<td>Gen4</td>
</tr>
</tbody>
</table>

A relation is a specific “table” consisting of:

- A name (e.g., VehicleType)
- Data values organized into rows and columns

A relation’s schema captures its “type” and consists of:

- The name of the relation (VehicleType)
- A name for each column called attributes (vt_id, make, model)
- A data type for each attribute called a “domain” (more later)
- Note: a relation fully conforms to its schema (structure)

Additional terminology

- Rows are also referred to as tuples or records
- Attributes are also referred to as fields
- A schema can denote an “entity” (e.g., Customer) where rows are “instances”
- A relation is also sometimes referred to as an “instance” of the schema

We often write a relation’s schema as $R(a_1, a_2, \ldots, a_n)$, for instance:

VehicleType(vt_id, make, model)
VehicleType

<table>
<thead>
<tr>
<th>vt_id</th>
<th>make</th>
<th>model</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Segway</td>
<td>Ninebot</td>
</tr>
<tr>
<td>2</td>
<td>Lime-S</td>
<td>Gen4</td>
</tr>
</tbody>
</table>

The **arity** (aka **degree**) of a relation is the number of attributes

- What is the arity of the example? ... 3
- How could we increase the arity of the example? ... add a column

The **cardinality** of a relation is the number of tuples

- What is the cardinality of this table? ... 2
- How could we increase the cardinality of the example? ... add a tuple

An attribute **domain** consists of both:

- the storage type (data type) of an attribute ... e.g., int, float, etc
- the set of allowable values of an attribute (constraint) ... e.g., age

We assume domains are **atomic**

- as opposed to collection-based
- non-atomic example: a **vehicles** attribute with a list of vehicle ids
- note that strings are considered atomic types

**NULL** is a special value ...

- usually means the value is unknown or doesn’t exist (not applicable)
- e.g., some vehicle types might not have an “**alternative_name**” value
A database is a collection of tables

- the database schema is the corresponding collection of relation schemas
- the database instance is the corresponding collection of relations
- DBMSs can manage many databases, each distinguished via a database name

Check In:

- What are the three main parts of a relation schema?
- What is meant by the arity of a relation?
- What is meant by the cardinality of a relation?
- Can the arity of a relation be determined by its schema?
- What are the two parts of a domain?
- What do we mean by an atomic domain?
- Give an example of an attribute where NULL means unknown.
- Give an example of an attribute where NULL means not-applicable.
**Key Constraints**

A key is a table attribute (or set of attributes) with unique values

- A key means each row in the table must have a unique key value
- Where a key value cannot be NULL
- We typically underline the key attribute(s) in a relation

Q: For the following, what are potential keys?

- `VehicleType(vt_id, make, model)`
- `Vehicle(v_id, vt_id, lat, lon)`
- `Trip(v_id, c_id, start_time, charge)`

**Simple versus composite keys**

- A simple key consists of a single attribute, e.g., `vt_id` and `v_id`
- A composite key consists of multiple attributes, e.g., `(v_id, start_time)`
- Note: we write composite keys using parens
More on Keys

Relations can have more than one set of key attributes:

\[ \text{Customer(c_id, email, ssn, fname, ...)} \]

- for example, this relation has three possible keys!
- each is a separate candidate key (vs a composite key)

Designating attributes as keys is a type of (operational) constraint

- DBMS will not allow duplicate row values to be inserted!
- we say the DBMS enforces a (designated) key constraint

A primary key is a special designated key in a DBMS

- the primary key is chosen by the database designer (physical design)
- DBMS enforces the primary key and uses it to optimize storage (more later)
- if multiple candidate keys, only one selected as primary
- other keys can be enforced by marking them as “unique” (constraint)
We are usually interested in **minimal** candidate keys

- minimal implies can’t remove an attribute from a key (and still have a key)
- this means all simple keys are minimal
- note that we can always add attributes to a key and still have a key

**Examples:**

**Vehicle**\((v\_id, vt\_id, lat, lon)\)

- \(v\_id\) is the primary key
- thus, each row has a unique \(v\_id\) value (key constraint)
- \(v\_id\) is a minimal key (since it is simple)
- \((v\_id, lat)\) is a non-minimal candidate key

**Trip**\((v\_id, start\_time, c\_id, charge)\)

- \((v\_id, start\_time)\) is the primary key and is minimal
- \((v\_id, start\_time, charge)\) is a non-minimal candidate key

**Check In:**

- What constraint is implied by a key?
- What is the difference between a simple and composite key?
- What is the difference between a composite key and a candidate key?
- What is a primary key?
- Given a table, identify the potential candidate keys