6.170: Software Studio
Relational Design: Normalization & Constraints
(some material adapted from Jennifer Widom's MOOC)

Arvind Satyanarayan +
Daniel Jackson
Monday

Ways of Persisting State
- Objects (a la Fritter)
- NoSQL (document stores)
- Relational Databases (SQL)

SQL
- Select Statements

```sql
select attr_1, attr_2, ..., attr_n
from rel_1, rel_2, ..., rel_n
where condition
```
Monday

Ways of Persisting State
Objects (a la Fritter)
NoSQL (document stores)
Relational Databases (SQL)

SQL
Select Statements
Subqueries

Subqueries in `from`

```
from rel₁, ..., (select attr₁, from rel₁, ..., where condition) as R
```

```
select attr₁, attr₂, ..., attrₙ
from rel₁, rel₂, ..., relₙ
where condition
```

Subqueries in `where`

```
where attr₁ in (select attr₁, ..., from rel₁, ... where condition)
```

```
where attr₁ exists (select attr₁, ..., from rel₁, ... where condition)
```
Monday

Ways of Persisting State
Objects (a la Fritter)
NoSQL (document stores)
Relational Databases (SQL)

SQL
Select Statements
Subqueries
Aggregation

Aggregation functions
min, max, sum, avg, count

\[
\text{select } \text{attr}_1, \text{attr}_2, \ldots, \text{attr}_n \\
\text{from } \text{rel}_1, \text{rel}_2, \ldots, \text{rel}_n \\
\text{where } \text{condition} \\
\text{group by } \text{attr}_1, \ldots, \text{attr}_n \\
\text{having } \text{condition}
\]

Post-aggregation filter
Monday

Ways of Persisting State
Objects (a la Fritter)
NoSQL (document stores)
Relational Databases (SQL)

SQL
Select Statements
Subqueries
Aggregation
Modification

\[
\begin{align*}
\text{insert into } & \text{ rel} \\
\text{values}(\text{attr}_1, \text{attr}_2, \ldots, \text{attr}_n) \\
\text{insert into } & \text{ rel} \\
\text{select-statement} \\
\text{update } & \text{ rel} \\
\text{set } & \text{attr}_1 = \text{expr}_1, \ldots, \text{attr}_n = \text{expr}_n \\
\text{where } & \text{condition} \\
\text{delete from } & \text{ rel} \\
\text{where } & \text{condition}
\end{align*}
\]
Monday

Ways of Persisting State
Objects (a la Fritter)
NoSQL (document stores)
Relational Databases (SQL)

SQL
Select Statements
Subqueries
Aggregation
Modification

Today

Separation of Concerns
Design Anomalies
Normalization Process
Robustness and Integrity via Constraints
Separation of Concerns

Columns are not *atomic*. Tuples can specify multiple values = opportunities for mistakes and inconsistencies.

<table>
<thead>
<tr>
<th>SSN</th>
<th>FullName</th>
<th>Colleges</th>
<th>HighSchool</th>
<th>HSCity</th>
<th>HSZip</th>
<th>Hobbies</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Anne Costa</td>
<td>MIT, Stanford</td>
<td>TJHSST</td>
<td>Alexandria</td>
<td>22312</td>
<td>Piano, Debate</td>
</tr>
<tr>
<td>876</td>
<td>Irene Adams</td>
<td>Michigan + UIUC</td>
<td>BAA</td>
<td>Dorchester</td>
<td>2122</td>
<td>Theater</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td></td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>
### Applications

<table>
<thead>
<tr>
<th>SSN</th>
<th>Fullname</th>
<th>College</th>
<th>HighSchool</th>
<th>HSCity</th>
<th>HSZip</th>
<th>Hobby</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Anne Costa</td>
<td>MIT</td>
<td>TJHSST</td>
<td>Alexandria</td>
<td>22312</td>
<td>Piano</td>
</tr>
<tr>
<td>123</td>
<td>Anne Costa</td>
<td>MIT</td>
<td>TJHSST</td>
<td>Alexandria</td>
<td>22312</td>
<td>Debate</td>
</tr>
<tr>
<td>123</td>
<td>Anne Costa</td>
<td>Stanford</td>
<td>TJHSST</td>
<td>Alexandria</td>
<td>22312</td>
<td>Piano</td>
</tr>
<tr>
<td>123</td>
<td>Anne Costa</td>
<td>Stanford</td>
<td>TJHSST</td>
<td>Alexandria</td>
<td>22312</td>
<td>Debate</td>
</tr>
<tr>
<td>876</td>
<td>Irene Adams</td>
<td>Michigan</td>
<td>BAA</td>
<td>Dorchester</td>
<td>2122</td>
<td>Theater</td>
</tr>
<tr>
<td>876</td>
<td>Irene Adams</td>
<td>UIUC</td>
<td>BAA</td>
<td>Dorchester</td>
<td>2122</td>
<td>Theater</td>
</tr>
</tbody>
</table>
Separation of Concerns: Design Anomalies

Applications

<table>
<thead>
<tr>
<th>SSN</th>
<th>FullName</th>
<th>College</th>
<th>HighSchool</th>
<th>HSCity</th>
<th>HSZip</th>
<th>Hobby</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Anne Costa</td>
<td>MIT</td>
<td>TJHSST</td>
<td>Alexandria</td>
<td>22312</td>
<td>Piano</td>
</tr>
<tr>
<td>123</td>
<td>Anne Costa</td>
<td>MIT</td>
<td>TJHSST</td>
<td>Alexandria</td>
<td>22312</td>
<td>Model UN</td>
</tr>
<tr>
<td>123</td>
<td>Anne Costa</td>
<td>Stanford</td>
<td>TJHSST</td>
<td>Alexandria</td>
<td>22312</td>
<td>Piano</td>
</tr>
<tr>
<td>123</td>
<td>Anne Costa</td>
<td>Stanford</td>
<td>TJHSST</td>
<td>Alexandria</td>
<td>22312</td>
<td>Debate</td>
</tr>
<tr>
<td>876</td>
<td>Irene Adams</td>
<td>Michigan</td>
<td>BAA</td>
<td>Dorchester</td>
<td>2122</td>
<td>Theater</td>
</tr>
<tr>
<td>876</td>
<td>Irene Adams</td>
<td>UIUC</td>
<td>BAA</td>
<td>Dorchester</td>
<td>2122</td>
<td>Theater</td>
</tr>
</tbody>
</table>

- **Redundancy**: Capturing the same information several times = storage space concerns, opportunities for mistakes.
- **Update Anomaly**: Facts may be updated inconsistently.
- **Delete Anomaly**: Knock-on effects such that more information is lost than expected.

DELETE FROM Applications WHERE hobby = 'Theater'

Inadvertently erases Irene Adams from existence!
Separation of Concerns: **Normalization**

"the key, 
the whole key, 
and nothing but the key, 
so help you Codd"
Separation of Concerns: *Normalization*

"the **key** (1NF),
the whole **key** (2NF),
and nothing but the **key** (3NF),
so help you Codd"

**Key**: attribute (or set of attributes) whose value is unique for every tuple.

**Primary Key**: a special key for identifying and referencing tuples.
Separation of Concerns: **Normalization**

"the **key** (1NF),
the whole **key** (2NF),
and nothing but the **key** (3NF),
so help you **Codd**"

**Key**: attribute (or set of attributes) whose value is unique for every tuple.

**Primary Key**: a special key for identifying and referencing tuples.

Ted Codd: invented relational model and theory for databases.
1NF: Establish the Key

1. *Atomic* columns: tuples may contain only one value per column.

### Applications

<table>
<thead>
<tr>
<th>SSN</th>
<th>FName</th>
<th>Colleges</th>
<th>Hobbies</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Anne Costa</td>
<td>MIT, Stanford</td>
<td>Piano, Debate</td>
</tr>
<tr>
<td>876</td>
<td>Irene Adams</td>
<td>Michigan + UIUC</td>
<td>Theater</td>
</tr>
</tbody>
</table>

### Table

<table>
<thead>
<tr>
<th>SSN</th>
<th>FName</th>
<th>College1</th>
<th>College2</th>
<th>Hobby1</th>
<th>Hobby2</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Anne Costa</td>
<td>MIT</td>
<td>Stanford</td>
<td>Piano</td>
<td>Debate</td>
</tr>
<tr>
<td>876</td>
<td>Irene Adams</td>
<td>Michigan</td>
<td>UIUC</td>
<td>Theater</td>
<td></td>
</tr>
</tbody>
</table>
1NF: Establish the Key

1. **Atomic** columns: tuples may contain only one value per column.

2. No **repeating groups** of columns: introduces arbitrary ordering, makes searching more difficult.

### Applications

<table>
<thead>
<tr>
<th>SSN</th>
<th>FullName</th>
<th>Colleges</th>
<th>Hobbies</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Anne Costa</td>
<td>MIT, Stanford</td>
<td>Piano, Debate</td>
</tr>
<tr>
<td>876</td>
<td>Irene Adams</td>
<td>Michigan + UIUC</td>
<td>Theater</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SSN</th>
<th>FullName</th>
<th>College1</th>
<th>College2</th>
<th>Hobby1</th>
<th>Hobby2</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Anne Costa</td>
<td>MIT</td>
<td>Stanford</td>
<td>Piano</td>
<td>Debate</td>
</tr>
<tr>
<td>876</td>
<td>Irene Adams</td>
<td>Michigan</td>
<td>UIUC</td>
<td>Theater</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SSN</th>
<th>FullName</th>
<th>College</th>
<th>Hobby</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Anne Costa</td>
<td>MIT</td>
<td>Piano</td>
</tr>
<tr>
<td>123</td>
<td>Anne Costa</td>
<td>MIT</td>
<td>Debate</td>
</tr>
<tr>
<td>123</td>
<td>Anne Costa</td>
<td>Stanford</td>
<td>Piano</td>
</tr>
<tr>
<td>123</td>
<td>Anne Costa</td>
<td>Stanford</td>
<td>Debate</td>
</tr>
</tbody>
</table>
1NF: Establish the Key

1. **Atomic** columns: tuples may contain only one value per column.

2. No **repeating groups** of columns: introduces arbitrary ordering, makes searching more difficult.

3. Create a **separate relation** for sets of related data, and identify each set with a **primary key**.

<table>
<thead>
<tr>
<th>SSN</th>
<th>FullName</th>
<th>Colleges</th>
<th>Hobbies</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Anne Costa</td>
<td>MIT, Stanford</td>
<td>Piano, Debate</td>
</tr>
<tr>
<td>876</td>
<td>Irene Adams</td>
<td>Michigan + UIUC</td>
<td>Theater</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SSN</th>
<th>FullName</th>
<th>College1</th>
<th>College2</th>
<th>Hobby1</th>
<th>Hobby2</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Anne Costa</td>
<td>MIT</td>
<td>Stanford</td>
<td>Piano</td>
<td>Debate</td>
</tr>
<tr>
<td>876</td>
<td>Irene Adams</td>
<td>Michigan</td>
<td>UIUC</td>
<td>Theater</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SSN</th>
<th>FullName</th>
<th>College</th>
<th>Hobby</th>
</tr>
</thead>
<tbody>
<tr>
<td>123</td>
<td>Anne Costa</td>
<td>MIT</td>
<td>Piano</td>
</tr>
<tr>
<td>123</td>
<td>Anne Costa</td>
<td>MIT</td>
<td>Debate</td>
</tr>
<tr>
<td>123</td>
<td>Anne Costa</td>
<td>Stanford</td>
<td>Piano</td>
</tr>
<tr>
<td>123</td>
<td>Anne Costa</td>
<td>Stanford</td>
<td>Debate</td>
</tr>
<tr>
<td>876</td>
<td>Irene Adams</td>
<td>Michigan</td>
<td>Theater</td>
</tr>
</tbody>
</table>
1NF: Establish the Key

1. Atomic columns: tuples may contain only one value per column.

2. No repeating groups of columns: introduces arbitrary ordering, makes searching more difficult.

3. Create a separate relation for sets of related data, and identify each set with a primary key.

Applications( SSN, fullName, college, hobby )

Students( SSN, firstName, lastName )

Hobbies( SSN, hobby )

Applications( SSN, college )
In-Class Activity: Practice 1NF

**Receipt**

| Customer        | Curran, Rajani | (217) 521-6910 |

**Purchases**

<table>
<thead>
<tr>
<th>Title</th>
<th>Type</th>
<th>Artist</th>
<th>Studio</th>
<th>Date</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Water Lilies</td>
<td>Oil Painting</td>
<td>Claude Monet</td>
<td>Studio 54</td>
<td>9/28/2018</td>
<td>$1,500,000</td>
</tr>
<tr>
<td>The Thinker</td>
<td>Bronze Sculpture</td>
<td>Auguste Rodin</td>
<td>L'atelier</td>
<td>07/30/1989</td>
<td>$10,000,000</td>
</tr>
<tr>
<td>Fishing Boats</td>
<td>Watercolor Painting</td>
<td>Vincent Van Gogh</td>
<td>Werkplaat</td>
<td>5/23/2001</td>
<td>$250,000</td>
</tr>
<tr>
<td>Starry Night</td>
<td>Oil Painting</td>
<td>Vincent Van Gogh</td>
<td>Werkplaat</td>
<td>11/29/1999</td>
<td>$4,800,000</td>
</tr>
</tbody>
</table>

1. In what ways does this schema violate 1NF?

2. Convert it to 1NF.

**Purchases**(Name, PhoneNumber, Titles, Types, Artists, Studios, Dates, Prices)
In-Class Activity: Practice 1NF

### Receipt

**Customer**
Curran, Rajani  
(217) 521-6910

**Purchases**

<table>
<thead>
<tr>
<th>Title</th>
<th>Type</th>
<th>Artist</th>
<th>Studio</th>
<th>Date</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Water Lilies</td>
<td>Oil Painting</td>
<td>Claude Monet</td>
<td>Studio 54</td>
<td>9/28/2018</td>
<td>$1,500,000</td>
</tr>
<tr>
<td>The Thinker</td>
<td>Bronze Sculpture</td>
<td>Auguste Rodin</td>
<td>L’atelier</td>
<td>07/30/1989</td>
<td>$10,000,000</td>
</tr>
<tr>
<td>Fishing Boats</td>
<td>Watercolor Painting</td>
<td>Vincent Van Gogh</td>
<td>Werkplaat</td>
<td>5/23/2001</td>
<td>$250,000</td>
</tr>
<tr>
<td>Starry Night</td>
<td>Oil Painting</td>
<td>Vincent Van Gogh</td>
<td>Werkplaat</td>
<td>11/29/1999</td>
<td>$4,800,000</td>
</tr>
</tbody>
</table>

**Customers**

- `cID`, `firstName`, `lastName`, `phoneNumber`

**Purchases**

- `cID`, `title`, `date`, `price`, `form`, `medium`, `artistFName`, `artistLName`, `studio`
2NF: The *whole* key

Students: $(SSN, firstName, lastName, highSchool, HSCity, HSZip)$

Hobbies: $(SSN, hobby)$

Applications: $(SSN, collegeName, collegeState)$

Why? 2NF violations introduce data redundancies (duplicating high school for every student, or college $\leftrightarrow$ state for every application).

No *partial dependencies*: all non-key columns dependent on the table's whole primary key. Or: Do all columns describe what the primary key uniquely identifies?

High school details are not dependent on (nor do they describe) a student's SSN. A college's state depends only on the college, not on the student applying.
2NF: The whole key

Fix violations by decomposing/extracting non-dependencies into new relations.

Students(\texttt{SSN, firstName, lastName, highSchool, HSCity, HSZip})
Hobbies(\texttt{SSN, hobby})
Applications(\texttt{SSN, collegeName, collegeState})

Students(\texttt{SSN, firstName, lastName, highSchool})
HighSchools(\texttt{highSchool, city, zip})
Hobbies(\texttt{SSN, hobby})
Colleges(\texttt{collegeName, collegeState})
Applications(\texttt{SSN, collegeName})

Intersection relation: tuples only hold references to tuples in other relations.
In-Class Activity: Practice 2NF

<table>
<thead>
<tr>
<th>Title</th>
<th>Type</th>
<th>Artist</th>
<th>Studio</th>
<th>Date</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Water Lilies</td>
<td>Oil Painting</td>
<td>Claude Monet</td>
<td>Studio 54</td>
<td>9/28/2018</td>
<td>$1,500,000</td>
</tr>
<tr>
<td>The Thinker</td>
<td>Bronze Sculpture</td>
<td>Auguste Rodin</td>
<td>L'atelier</td>
<td>07/30/1989</td>
<td>$10,000,000</td>
</tr>
<tr>
<td>Fishing Boats</td>
<td>Watercolor Painting</td>
<td>Vincent Van Gogh</td>
<td>Werkplaat</td>
<td>5/23/2001</td>
<td>$250,000</td>
</tr>
<tr>
<td>Starry Night</td>
<td>Oil Painting</td>
<td>Vincent Van Gogh</td>
<td>Werkplaat</td>
<td>11/29/1999</td>
<td>$4,800,000</td>
</tr>
</tbody>
</table>

1. In what ways does this schema violate 2NF?

Customers(\textit{cID}, firstName, lastName, phoneNumber)

Purchases(\textit{cID}, title, date, price, form, medium, artistFName, artistLName, studio)

2. Convert it to 2NF.
In-Class Activity: Practice 2NF

**Receipt**

**Customer**
Curran, Rajani
(217) 521-6910

**Purchases**

<table>
<thead>
<tr>
<th>Title</th>
<th>Type</th>
<th>Artist</th>
<th>Studio</th>
<th>Date</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Water Lilies</td>
<td>Oil Painting</td>
<td>Claude Monet</td>
<td>Studio 54</td>
<td>9/28/2018</td>
<td>$1,500,000</td>
</tr>
<tr>
<td>The Thinker</td>
<td>Bronze Sculpture</td>
<td>Auguste Rodin</td>
<td>L’atelier</td>
<td>07/30/1989</td>
<td>$10,000,000</td>
</tr>
<tr>
<td>Fishing Boats</td>
<td>Watercolor Painting</td>
<td>Vincent Van Gogh</td>
<td>Werkplaat</td>
<td>5/23/2001</td>
<td>$250,000</td>
</tr>
<tr>
<td>Starry Night</td>
<td>Oil Painting</td>
<td>Vincent Van Gogh</td>
<td>Werkplaat</td>
<td>11/29/1999</td>
<td>$4,800,000</td>
</tr>
</tbody>
</table>

**Customers**
\(cID, \text{firstName}, \text{lastName}, \text{phoneNumber}\)

**Purchases**
\(cID, \text{title}, \text{date}, \text{price}, \text{form}, \text{medium}, \text{artistFName}, \text{artistLName}, \text{studio}\)

**Customers**
\(cID, \text{firstName}, \text{lastName}, \text{phoneNumber}\)

**Purchases**
\(cID, \text{artID}, \text{date}, \text{price}\)

**Art**
\(\text{artID}, \text{title}, \text{form}, \text{medium}, \text{artistFName}, \text{artistLName}, \text{studio}\)
3NF violations introduce opportunities for update anomalies (i.e., updating a city or zip, and forgetting to update the other).

No *transitive dependencies*: do non-key columns depend on *other* non-key columns? or Can the value of a column be derived from (or describe) another column?

Students (**SSN**, **firstName**, **lastName**, **highSchool**)
HighSchools (**highSchool**, **city**, **zip**)
Hobbies (**SSN**, **hobby**)
Colleges (**collegeName**, **collegeState**)
Applications (**SSN**, **collegeName**)

A high school's city could be derived from the zip code.
3NF: Nothing \textit{but the key}

Fix violations by decomposing/extracting transitive dependencies into \textit{new relations}.

\begin{itemize}
  \item Students(\textit{SSN}, \textit{firstName}, \textit{lastName}, \textit{highSchool})
  \item HighSchools(\textit{highSchool}, \textit{zip})
  \item Locations(\textit{zip}, \textit{city})
  \item Hobbies(\textit{SSN}, \textit{hobby})
  \item Colleges(\textit{collegeName}, \textit{collegeState})
  \item Applications(\textit{SSN}, \textit{collegeName})
\end{itemize}
Fix violations by decomposing/extracting transitive dependencies into new relations.

Students(\texttt{SSN}, \texttt{firstName}, \texttt{lastName}, \texttt{highSchool})
HighSchools(\texttt{highSchool}, \texttt{zip})
Locations(\texttt{zip}, \texttt{city}, \texttt{state})
Hobbies(\texttt{SSN}, \texttt{hobby})
Colleges(\texttt{collegeName}, \texttt{zip})
Applications(\texttt{SSN}, \texttt{collegeName})

Can you over-normalize a database?

Yes! Separating concerns is good, but join operations (querying across multiple relations) can be slow.

\textit{Intensional denormalization} reintroduces redundancies to speed up reads (at the expense of writes/updates).

May be easier to guard against update anomalies on the client side.
In-Class Activity: Practice 3NF

1. In what ways does this schema violate 3NF?

Customers(\(c\text{ID}, \text{firstName}, \text{lastName}, \text{phoneNumber}\))

Purchases(\(c\text{ID}, \text{artID}, \text{date}, \text{price}\))

Art(\(\text{artID}, \text{title}, \text{form}, \text{medium}, \text{artistFName}, \text{artistLName}, \text{studio}\))

2. Convert it to 3NF.
In-Class Activity: Practice 3NF

## Receipt

### Customer
Curran, Rajani  
(217) 521-6910

### Purchases

<table>
<thead>
<tr>
<th>Title</th>
<th>Type</th>
<th>Artist</th>
<th>Studio</th>
<th>Date</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>Water Lilies</td>
<td>Oil Painting</td>
<td>Claude Monet</td>
<td>Studio 54</td>
<td>9/28/2018</td>
<td>$1,500,000</td>
</tr>
<tr>
<td>The Thinker</td>
<td>Bronze Sculpture</td>
<td>Auguste Rodin</td>
<td>L'atelier</td>
<td>07/30/1989</td>
<td>$10,000,000</td>
</tr>
<tr>
<td>Fishing Boats</td>
<td>Watercolor Painting</td>
<td>Vincent Van Gogh</td>
<td>Werkplaat</td>
<td>5/23/2001</td>
<td>$250,000</td>
</tr>
<tr>
<td>Starry Night</td>
<td>Oil Painting</td>
<td>Vincent Van Gogh</td>
<td>Werkplaat</td>
<td>11/29/1999</td>
<td>$4,800,000</td>
</tr>
</tbody>
</table>

Customers(\textit{cID}, \textit{firstName}, \textit{lastName}, \textit{phoneNumber})

Purchases(\textit{cID}, \textit{artID}, \textit{date}, \textit{price})

Art(\textit{artID}, \textit{title}, \textit{form}, \textit{medium}, \textit{artistID})

Artists(\textit{artistID}, \textit{firstName}, \textit{lastName}, \textit{studio})
Attributes and schema normalization already impose constraints on the type and structure of the data in our database.

**Integrity Constraints**

Impose *semantic* constraints that have to do with our particular application.

\[ 0.0 \leq \text{GPA} \leq 4.0 \]

\[ \text{state} \in [\text{'AL', 'AK', \ldots, 'WY'}] \]
Integrity Constraints

Impose semantic constraints that have to do with our particular application.

✓ Catch data entry errors (i.e., on insert).
✓ Catch correctness errors (i.e., on update).
✓ Enforce consistency between data (e.g., between tuple references across tables).
✓ More efficient storage + performance.

0.0 ≤ GPA ≤ 4.0
state ∈ ['AL', 'AK', ... ' WY']
Integrity Constraints

Not null.

create table Students (  
  SSN int,  
  firstName text,  
  lastName text not null,  
  highSchool text  
);
Integrity Constraints

Not null.

Keys (enforce uniqueness). Primary keys cannot be null.

create table Students (  
  SSN int primary key,  
  firstName text unique,  
  lastName text not null,  
  highSchool text  
);

create table Colleges (  
  name text,  
  state text,  
  primary key (name, state)  
);

create table Applications (  
  appID int primary key,  
  ...  
  unique (SSN, college),  
  unique (SSN, major),  
);
Integrity Constraints

Not null.

Keys (enforce uniqueness). Primary keys cannot be null.

Foreign key (referential integrity).

```sql
create table Students (  
    SSN int primary key,  
    firstName text unique,  
    lastName text not null,  
    highSchool text  
);

create table Colleges (  
    name text,  
    state text,  
    primary key (name, state)  
);

create table Applications (  
    appID int primary key,  
    SSN int references Students(SSN),  
    college text references Colleges(name)  
    ...  
);
```
Integrity Constraints

Not null.

Key (enforce uniqueness).

Foreign key (referential integrity).

Attribute-based.

Tuple-based.

create table Students (  
  SSN int primary key,  
  firstName text,  
  lastName text not null,  
  highSchool text,  
  GPA decimal(2, 1)  
    check(GPA <= 4.0 and GPA > 0.0)  
);
In-Class Activity: Constraints

Write the schema definition (i.e., create table statements) for our art gallery. Include constraints for:

- all primary keys
- all foreign keys
- at least one attribute/tuple/not-null constraint.

Customers(\texttt{cID}, \texttt{firstName}, \texttt{lastName}, \texttt{phoneNumber})

Purchases(\texttt{cID}, \texttt{artID}, \texttt{date}, \texttt{price})

Art(\texttt{artID}, \texttt{title}, \texttt{form}, \texttt{medium}, \texttt{artistID})

Artists(\texttt{artistID}, \texttt{firstName}, \texttt{lastName}, \texttt{studio})
In-Class Activity: Constraints

create table Customers (  
cID int primary key,  
firstName text,  
lastName text,  
phoneNumber int 
);

create table Artists (  
artistID int primary key,  
firstName text,  
lastName text,  
studio text  
);

create table Art (  
artId int primary key,  
title text,  
form text,  
medium text,  
artistID int references Artists(artistID) 
);

create table Purchases (  
cID int references Customers(cID),  
artID int references Art(artID),  
date date,  
price float check(price >= 100),  
primary key(cID, artID, date) 
);