Part II

Relational Databases – Data as Tables
Educational Objective for Today . . .

- Basic understanding of the structure of relational databases
Educational Objective for Today . . .

- Basic understanding of the structure of relational databases
- Knowledge of base operations of relational query languages
Educational Objective for Today . . .

- Basic understanding of the structure of relational databases
- Knowledge of base operations of relational query languages
- Elementary ability to use SQL
Relational Model

- Conceptually, a database is a set of tables

<table>
<thead>
<tr>
<th>WINES</th>
<th>WineID</th>
<th>Name</th>
<th>Color</th>
<th>Vintage</th>
<th>Vineyard</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1042</td>
<td>La Rose Grand Cru</td>
<td>Red</td>
<td>1998</td>
<td>Château La Rose</td>
</tr>
<tr>
<td></td>
<td>2168</td>
<td>Creek Shiraz</td>
<td>Red</td>
<td>2003</td>
<td>Creek</td>
</tr>
<tr>
<td></td>
<td>3456</td>
<td>Zinfandel</td>
<td>Red</td>
<td>2004</td>
<td>Helena</td>
</tr>
<tr>
<td></td>
<td>2171</td>
<td>Pinot Noir</td>
<td>Red</td>
<td>2001</td>
<td>Creek</td>
</tr>
<tr>
<td></td>
<td>3478</td>
<td>Pinot Noir</td>
<td>Red</td>
<td>1999</td>
<td>Helena</td>
</tr>
<tr>
<td></td>
<td>4711</td>
<td>Riesling Reserve</td>
<td>White</td>
<td>1999</td>
<td>Müller</td>
</tr>
<tr>
<td></td>
<td>4961</td>
<td>Chardonnay</td>
<td>White</td>
<td>2002</td>
<td>Bighorn</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ORIGIN</th>
<th>Vineyard</th>
<th>District</th>
<th>Region</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Creek</td>
<td>Barossa Valley</td>
<td>South Australia</td>
</tr>
<tr>
<td></td>
<td>Helena</td>
<td>Napa Valley</td>
<td>California</td>
</tr>
<tr>
<td></td>
<td>Château La Rose</td>
<td>Saint-Emilion</td>
<td>Bordeaux</td>
</tr>
<tr>
<td></td>
<td>Château La Pointe</td>
<td>Pomerol</td>
<td>Bordeaux</td>
</tr>
<tr>
<td></td>
<td>Müller</td>
<td>Rheingau</td>
<td>Hessen</td>
</tr>
<tr>
<td></td>
<td>Bighorn</td>
<td>Napa Valley</td>
<td>California</td>
</tr>
</tbody>
</table>

- Table = “Relation”
Presentation of Relations; Terminology

- **Bold** lines: relation schema
- Further entries in the table: relation
- A table row: tuple
- A column heading: attribute
- An entry: attribute value
Integrity Constraints: Keys

- Attributes of a column unambiguously identify saved tuples: key property
- E.g., Vineyard for table ORIGIN

<table>
<thead>
<tr>
<th>ORIGIN</th>
<th>Vineyard</th>
<th>District</th>
<th>Region</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creek</td>
<td>Barossa Valley</td>
<td>South Australia</td>
<td></td>
</tr>
<tr>
<td>Helena</td>
<td>Napa Valley</td>
<td>California</td>
<td></td>
</tr>
<tr>
<td>Château La Rose</td>
<td>Saint-Emilion</td>
<td>Bordeaux</td>
<td></td>
</tr>
<tr>
<td>Château La Pointe</td>
<td>Pomerol</td>
<td>Bordeaux</td>
<td></td>
</tr>
<tr>
<td>Müller</td>
<td>Rheingau</td>
<td>Hessen</td>
<td></td>
</tr>
<tr>
<td>Bighorn</td>
<td>Napa Valley</td>
<td>California</td>
<td></td>
</tr>
</tbody>
</table>

- Combinations of attributes can also be keys!
- Keys can be marked by underlining them
Integrity Constraints: Foreign Keys

- Keys in one table can be used as unambiguous references in another table (or even in the same table!): Foreign key, referential integrity
- E.g., Vineyard as a reference to ORIGIN
- A foreign key is a key in a “foreign” table
# Foreign Keys /2

<table>
<thead>
<tr>
<th>WINES</th>
<th>WineID</th>
<th>Name</th>
<th>Color</th>
<th>Vintage</th>
<th>Vineyard → ORIGIN</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>1042</td>
<td>La Rose Grand Cru</td>
<td>Red</td>
<td>1998</td>
<td>Château La Rose</td>
</tr>
<tr>
<td></td>
<td>2168</td>
<td>Creek Shiraz</td>
<td>Red</td>
<td>2003</td>
<td>Creek</td>
</tr>
<tr>
<td></td>
<td>3456</td>
<td>Zinfandel</td>
<td>Red</td>
<td>2004</td>
<td>Helena</td>
</tr>
<tr>
<td></td>
<td>2171</td>
<td>Pinot Noir</td>
<td>Red</td>
<td>2001</td>
<td>Creek</td>
</tr>
<tr>
<td></td>
<td>3478</td>
<td>Pinot Noir</td>
<td>Red</td>
<td>1999</td>
<td>Helena</td>
</tr>
<tr>
<td></td>
<td>4711</td>
<td>Riesling Reserve</td>
<td>White</td>
<td>1999</td>
<td>Müller</td>
</tr>
<tr>
<td></td>
<td>4961</td>
<td>Chardonnay</td>
<td>White</td>
<td>2002</td>
<td>Bighorn</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ORIGIN</th>
<th>Vineyard</th>
<th>District</th>
<th>Region</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Creek</td>
<td>Barossa Valley</td>
<td>South Australia</td>
</tr>
<tr>
<td></td>
<td>Helena</td>
<td>Napa Valley</td>
<td>California</td>
</tr>
<tr>
<td></td>
<td>Château La Rose</td>
<td>Saint-Emilion</td>
<td>Bordeaux</td>
</tr>
<tr>
<td></td>
<td>Château La Pointe</td>
<td>Pomerol</td>
<td>Bordeaux</td>
</tr>
<tr>
<td></td>
<td>Müller</td>
<td>Rheingau</td>
<td>Hessen</td>
</tr>
<tr>
<td></td>
<td>Bighorn</td>
<td>Napa Valley</td>
<td>California</td>
</tr>
</tbody>
</table>
The `create table` Statement

```
create table base_relation_name (  
column_name_1 domain_1 [not null],  
...  
column_name_k domain_k [not null])
```

- Effect of this command is both
  - to store the `relation schema` in the data dictionary, and
  - to prepare an “empty base relation” in the database
Possible Domains in SQL

- **integer** *(also: integer4, int),*
- **smallint** *(also: integer2),*
- **float(p)** *(also, for short, float),*
- **decimal(p,q)** and **numeric(p,q)** with **q** decimal places,
- **character(n)** *(also, for short, char(n), with n = 1 just char) for character strings of fixed length n,*
- **character varying(n)** *(also, for short, varchar(n) for variable-length character strings up to the maximum length n,*
- **bit(n)** or **bit varying(n)** like **varchar** but for bit strings, and
- **date, time, timestamp** for specifying dates, times and the combination of date and time
Example for `create table`:

```sql
create table WINES (
    WineID int not null,
    Name varchar(20) not null,
    Color varchar(10),
    Vintage int,
    Vineyard varchar(20),
    primary key(WineID))
```

- `primary key` marks column as `key attribute`
create table WINES (  
WineID int,  
Name varchar(20) not null,  
Color varchar(10),  
Vintage int,  
Vineyard varchar(20),  
primary key(WineID),  
foreign key(Vineyard)  
    references ORIGIN(Vineyard))

- foreign key marks column as a foreign key
Null Values

- **not null** precludes null values as attribute values for certain columns
- SQL uses **null** to refer to null values; we use ⊥
- **null** has the semantics of "unknown value", "value does not apply" oder "value does not exist"; however, **null** itself does not belong to any domain
- **null** can occur in any column, except for key attributes or columns marked **not null**
Additional Notes on Data Definition in SQL

- Apart from primary and foreign keys, SQL allows specifying the following:
  - Default values for attributes using the `default` clause,
  - `create domain` statement to define custom domains (data types), and
  - `check` clause to specify further local integrity constraints within the domains, attributes and relation schemata being defined.
Query Operations on Tables

- **Basic operations** on tables that allow computing new result tables from saved database tables
- Operations are combined to form the so-called relational algebra
- Mathematics: algebra is defined by a domain and operations defined on that domain
  → for database queries, the contents of the database are the values (of the domain), operations are functions to compute query results
- Query operations can be freely combined and form an algebra to perform “calculations on tables” – the so-called relational algebra
Relational Algebra: Overview

Selection

Projection

Join

Selection

Projection

Join

a1 b2
a2 b2
b2 c3
b3 c4
a2 b3
b4 c5
a1 b2
a2 b2
a2 b3
c3
c3
c4
Relational Databases – Data as Tables

Selection \( \sigma \)

- **Selection**: Choose rows in a table based on a selection predicate

\[ \sigma_{\text{Vintage} > 2000}(\text{WINES}) \]

<table>
<thead>
<tr>
<th>WineID</th>
<th>Name</th>
<th>Color</th>
<th>Vintage</th>
<th>Vineyard</th>
</tr>
</thead>
<tbody>
<tr>
<td>2168</td>
<td>Creek Shiraz</td>
<td>Red</td>
<td>2003</td>
<td>Creek</td>
</tr>
<tr>
<td>3456</td>
<td>Zinfandel</td>
<td>Red</td>
<td>2004</td>
<td>Helena</td>
</tr>
<tr>
<td>2171</td>
<td>Pinot Noir</td>
<td>Red</td>
<td>2001</td>
<td>Creek</td>
</tr>
<tr>
<td>4961</td>
<td>Chardonnay</td>
<td>White</td>
<td>2002</td>
<td>Bighorn</td>
</tr>
</tbody>
</table>
Projection $\pi$

- **Projection**: Choose columns by specifying a list of attributes

$$\pi_{\text{Region}}(\text{ORIGIN})$$

<table>
<thead>
<tr>
<th>Region</th>
</tr>
</thead>
<tbody>
<tr>
<td>South Australia</td>
</tr>
<tr>
<td>California</td>
</tr>
<tr>
<td>Bordeaux</td>
</tr>
<tr>
<td>Hessen</td>
</tr>
</tbody>
</table>
Projection $\pi$

- **Projection**: Choose columns by specifying a list of attributes

$$\pi_{\text{Region}}(\text{ORIGIN})$$

<table>
<thead>
<tr>
<th>Region</th>
</tr>
</thead>
<tbody>
<tr>
<td>South Australia</td>
</tr>
<tr>
<td>California</td>
</tr>
<tr>
<td>Bordeaux</td>
</tr>
<tr>
<td>Hessen</td>
</tr>
</tbody>
</table>

- Projection removes duplicate tuples.
**Natural Join**

- **Join**: connects tables via *same-named columns*, combining two tuples if they have *equal values* in those columns

WINES ⋊ ORIGIN

<table>
<thead>
<tr>
<th>WineID</th>
<th>Name</th>
<th>...</th>
<th>Vineyard</th>
<th>District</th>
<th>Region</th>
</tr>
</thead>
<tbody>
<tr>
<td>1042</td>
<td>La Rose Grand Cru</td>
<td>...</td>
<td>Ch. La Rose</td>
<td>Saint-Emilion</td>
<td>Bordeaux</td>
</tr>
<tr>
<td>2168</td>
<td>Creek Shiraz</td>
<td>...</td>
<td>Creek</td>
<td>Barossa Valley</td>
<td>South Australia</td>
</tr>
<tr>
<td>3456</td>
<td>Zinfandel</td>
<td>...</td>
<td>Helena</td>
<td>Napa Valley</td>
<td>California</td>
</tr>
<tr>
<td>2171</td>
<td>Pinot Noir</td>
<td>...</td>
<td>Creek</td>
<td>Barossa Valley</td>
<td>South Australia</td>
</tr>
<tr>
<td>3478</td>
<td>Pinot Noir</td>
<td>...</td>
<td>Helena</td>
<td>Napa Valley</td>
<td>California</td>
</tr>
<tr>
<td>4711</td>
<td>Riesling Reserve</td>
<td>...</td>
<td>Müller</td>
<td>Rheingau</td>
<td>Hessen</td>
</tr>
<tr>
<td>4961</td>
<td>Chardonnay</td>
<td>...</td>
<td>Bighorn</td>
<td>Napa Valley</td>
<td>California</td>
</tr>
</tbody>
</table>
Natural Join ▷

- **Join**: connects tables via *same-named columns*, combining two tuples if they have *equal values* in those columns.

**WINES ▷ ORIGIN**

<table>
<thead>
<tr>
<th>WineID</th>
<th>Name</th>
<th>...</th>
<th>Vineyard</th>
<th>District</th>
<th>Region</th>
</tr>
</thead>
<tbody>
<tr>
<td>1042</td>
<td>La Rose Grand Cru</td>
<td>...</td>
<td>Ch. La Rose</td>
<td>Saint-Emilion</td>
<td>Bordeaux</td>
</tr>
<tr>
<td>2168</td>
<td>Creek Shiraz</td>
<td>...</td>
<td>Creek</td>
<td>Barossa Valley</td>
<td>South Australia</td>
</tr>
<tr>
<td>3456</td>
<td>Zinfandel</td>
<td>...</td>
<td>Helena</td>
<td>Napa Valley</td>
<td>California</td>
</tr>
<tr>
<td>2171</td>
<td>Pinot Noir</td>
<td>...</td>
<td>Creek</td>
<td>Barossa Valley</td>
<td>South Australia</td>
</tr>
<tr>
<td>3478</td>
<td>Pinot Noir</td>
<td>...</td>
<td>Helena</td>
<td>Napa Valley</td>
<td>California</td>
</tr>
<tr>
<td>4711</td>
<td>Riesling Reserve</td>
<td>...</td>
<td>Müller</td>
<td>Rheingau</td>
<td>Hessen</td>
</tr>
<tr>
<td>4961</td>
<td>Chardonnay</td>
<td>...</td>
<td>Bighorn</td>
<td>Napa Valley</td>
<td>California</td>
</tr>
</tbody>
</table>

- The vineyard “Château La Pointe” is missing from the result due to tuples that do not find a partner (*dangling tuples*), are eliminated.
Combining Operations

\[ \pi\text{Name,Color,Vineyard} (\sigma\text{Vintage}>2000(WINES) \bowtie \sigma\text{Region='California'}(ORIGIN)) \]

yields

<table>
<thead>
<tr>
<th>Name</th>
<th>Color</th>
<th>Vineyard</th>
</tr>
</thead>
<tbody>
<tr>
<td>Zinfandel</td>
<td>Red</td>
<td>Helena</td>
</tr>
<tr>
<td>Chardonnay</td>
<td>White</td>
<td>Bighorn</td>
</tr>
</tbody>
</table>
Renaming $\beta$

Renaming to adapt attribute names:

<table>
<thead>
<tr>
<th>WINELIST</th>
<th>Name</th>
<th>RECOMMENDATION</th>
<th>Wine</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>La Rose Grand Cru</td>
<td></td>
<td>La Rose Grand Cru</td>
</tr>
<tr>
<td></td>
<td>Creek Shiraz</td>
<td></td>
<td>Riesling Reserve</td>
</tr>
<tr>
<td></td>
<td>Zinfandel</td>
<td></td>
<td>Merlot Selection</td>
</tr>
<tr>
<td></td>
<td>Pinot Noir</td>
<td></td>
<td>Sauvignon Blanc</td>
</tr>
<tr>
<td></td>
<td>Riesling Reserve</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Adapt with:

\[ \beta \text{Name} \leftrightarrow \text{Wine} \ (\text{RECOMMENDATION}) \]
Set Operations

- **Union** \( r_1 \cup r_2 \) of two relations \( r_1 \) and \( r_2 \): collects the tuple sets of two relations in a common schema.
- Both relations must have the same set of attributes.

\[
\text{WINELIST} \cup \beta_{\text{Name} \leftarrow \text{Wine}}(\text{RECOMMENDATION})
\]

<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>La Rose Grand Cru</td>
</tr>
<tr>
<td>Creek Shiraz</td>
</tr>
<tr>
<td>Zinfandel</td>
</tr>
<tr>
<td>Pinot Noir</td>
</tr>
<tr>
<td>Riesling Reserve</td>
</tr>
<tr>
<td>Merlot Selection</td>
</tr>
<tr>
<td>Sauvignon Blanc</td>
</tr>
</tbody>
</table>
Set Operations /2

- **Difference** \( r_1 - r_2 \) removes from the first relation all tuples that are present in the second relation

\[
\text{WINELIST} - \beta_{\text{Name} \leftarrow \text{Wine}}(\text{RECOMMENDATION})
\]

yields:

<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Creek Shiraz</td>
</tr>
<tr>
<td>Zinfandel</td>
</tr>
<tr>
<td>Pinot Noir</td>
</tr>
</tbody>
</table>
Set Operations /3

- **Intersection** $r_1 \cap r_2$: yields all tuples that are present in both relations

$$\text{WINELIST} \cap \beta_{\text{Name} \leftarrow \text{Wine}}(\text{RECOMMENDATION})$$

yields:

<table>
<thead>
<tr>
<th>Name</th>
</tr>
</thead>
<tbody>
<tr>
<td>La Rose Grand Cru</td>
</tr>
<tr>
<td>Riesling Reserve</td>
</tr>
</tbody>
</table>
SQL Query as a Standard Language

- Query a single table

```sql
select Name, Color
from WINES
where Vintage = 2002
```

- SQL has **multi-set semantics** — SQL does not automatically suppress duplicate table entries!
- Set semantics by using `distinct`

```sql
select distinct Name
from WINES
```
Joining Tables

- Cross join as basic join

```sql
select *
from WINES, ORIGIN
```

- Join with operator `natural join`

```sql
select *
from WINES natural join ORIGIN
```

- Alternatively, join by specifying a `join condition`

```sql
select *
from WINES, ORIGIN
where WINES.Vineyard = ORIGIN.Vineyard
```
Combining Conditions

- Expression in relational algebra

\[ \pi_{\text{Name,Color,Vineyard}}(\sigma_{\text{Vintage}>2000}(\text{WINES}) \Join \sigma_{\text{Region='California'}}(\text{ORIGIN})) \]

- Query in SQL

```sql
select Name, Color, WINES.Vineyard
from WINES, ORIGIN
where Vintage > 2000 and
    Region = 'California' and
    WINES.Vineyard = ORIGIN.Vineyard
```
Set Operations in SQL

- In SQL, union is realized by an extra operator, **union**
- Differences by using nested queries

```sql
select *
from WINEMAKER
where Name not in (  
    select Surname  
    from CRITIC)  
```
Manipulation Operations in SQL

- **insert**: Insert one or more tuples into a base relation or view
- **update**: Change one or more tuples in a base relation or view
- **delete**: Delete one or more tuples from a base relation or view

Local and global integrity constraints must be checked automatically by the system when executing manipulation operations.
The **update** Statement

- **Syntax:**

```plaintext
update base_relation
set attribute_1 = expression_1
...
attribute_n = expression_n
[ where condition ]
```
Example for **update**

<table>
<thead>
<tr>
<th>WineID</th>
<th>Name</th>
<th>Color</th>
<th>Vintage</th>
<th>Vineyard</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>2168</td>
<td>Creek Shiraz</td>
<td>Red</td>
<td>2003</td>
<td>Creek</td>
<td>7.99</td>
</tr>
<tr>
<td>3456</td>
<td>Zinfandel</td>
<td>Red</td>
<td>2004</td>
<td>Helena</td>
<td>5.99</td>
</tr>
<tr>
<td>2171</td>
<td>Pinot Noir</td>
<td>Red</td>
<td>2001</td>
<td>Creek</td>
<td>10.99</td>
</tr>
<tr>
<td>3478</td>
<td>Pinot Noir</td>
<td>Red</td>
<td>1999</td>
<td>Helena</td>
<td>19.99</td>
</tr>
<tr>
<td>4711</td>
<td>Riesling Reserve</td>
<td>White</td>
<td>1999</td>
<td>Müller</td>
<td>14.99</td>
</tr>
<tr>
<td>4961</td>
<td>Chardonnay</td>
<td>White</td>
<td>2002</td>
<td>Bighorn</td>
<td>9.90</td>
</tr>
</tbody>
</table>

**update** WINES
**set** Price = Price * 1.10
**where** Vintage < 2000
## Example for **update**: New Values

<table>
<thead>
<tr>
<th>WineID</th>
<th>Name</th>
<th>Color</th>
<th>Vintage</th>
<th>Vineyard</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>2168</td>
<td>Creek Shiraz</td>
<td>Red</td>
<td>2003</td>
<td>Creek</td>
<td>7.99</td>
</tr>
<tr>
<td>3456</td>
<td>Zinfandel</td>
<td>Red</td>
<td>2004</td>
<td>Helena</td>
<td>5.99</td>
</tr>
<tr>
<td>2171</td>
<td>Pinot Noir</td>
<td>Red</td>
<td>2001</td>
<td>Creek</td>
<td>10.99</td>
</tr>
<tr>
<td>3478</td>
<td>Pinot Noir</td>
<td>Red</td>
<td>1999</td>
<td>Helena</td>
<td>21.99</td>
</tr>
<tr>
<td>4711</td>
<td>Riesling Reserve</td>
<td>White</td>
<td>1999</td>
<td>Müller</td>
<td>16.49</td>
</tr>
<tr>
<td>4961</td>
<td>Chardonnay</td>
<td>White</td>
<td>2002</td>
<td>Bighorn</td>
<td>9.90</td>
</tr>
</tbody>
</table>
Additional Notes on update

- Operations on single tuples can be achieved by using the primary key:

  ```sql
  update WINES
  set Price = 7.99
  where WineID = 3456
  ```

- Update the whole relation:

  ```sql
  update WINES
  set Price = 11
  ```
The **delete** Statement

- **Syntax:**

  \[
  \text{delete} \\
  \text{from} \ \text{base\_relation} \\
  [ \ \text{where} \ \text{condition} \ ]
  \]

- Delete a tuple from the WINES relation:

  \[
  \text{delete from WINES} \\
  \text{where WineID = 4711}
  \]
Additional Notes on `delete`

- Deletion of multiple tuples is the common case:
  
  ```sql
  delete from WINES
  where Color = 'White'
  ```

- Delete the whole relation:
  
  ```sql
  delete from WINES
  ```
Additional Notes on `delete` /2

- Deletions can lead to violation of integrity constraints!
- Example: Violation of the foreign key property if there are still wines from this origin:

```
delete from ORIGIN
where District = 'Hessen'
```
The **insert** Statement

- **Syntax:**

```
insert
into  base_relation
    [ (attribute$_{1}$, ..., attribute$_{n}$) ]
values (constant$_{1}$, ..., constant$_{n}$)
```

- Optional list of attributes allows for insertion of incomplete tuples
**insert Examples**

```sql
insert into ORIGIN (Vineyard, Region)
values ('Wairau Hills', 'Marlborough')
```

- Not all attributes given → Value of missing attribute District will be null

```sql
insert into ORIGIN
values ('Château Lafite', 'Medoc', 'Bordeaux')
```
Inserting Computed Data

- Syntax:

```sql
insert
into  base_relation
[  (attribute_1, ..., attribute_n) ]
SQL-query
```

- Example:

```sql
insert into WINES (  
    select ProdID, ProdName, 'Red', ProdYear,  
          'Château Lafite'  
    from SUPPLIER  
    where SName = 'Aspri Spirits' )
```
Summary

- Relational model: database as a set of tables
- Integrity constraints in the relational model
- Table definition in SQL
- Relational algebra: query operators
- Basic concepts of SQL queries and manipulations
Control Questions

What is a relation?
Control Questions

- What is a relation?
- What are the defining properties of the relational algebra?
Control Questions

- What is a relation?
- What are the defining properties of the relational algebra?
- How are objects from the real world represented in a relational database?
Control Questions

- What is a relation?
- What are the defining properties of the relational algebra?
- How are objects from the real world represented in a relational database?
- How can tables in SQL be defined and manipulated?
Control Questions

- What is a relation?
- What are the defining properties of the relational algebra?
- How are objects from the real world represented in a relational database?
- How can tables in SQL be defined and manipulated?
- What are integrity constraints?