Topics

- Structured Query Language (SQL)
  - Data Definition Language (DDL)
  - Data Manipulation Language (DML)
    - Select
      - from, where
      - order by
      - set operations
      - joins (implicit, explicit)
      - sub-queries
      - aggregation and grouping
    - Insert
    - Delete
    - Update
SQL: Inserting Tuples

- **Relational Algebra Syntax:** \[ R \leftarrow R \cup E \]
  - Example:

  \[
  \text{passengers} \leftarrow \text{passengers} \cup \{(\text{‘David’}, \text{NULL}, \text{‘Chiu’}, \text{‘888-88-8888’})\}
  \]

- **SQL Syntax:**
  
  ```sql
  INSERT INTO R VALUES (v1, v2, ..., vn);
  
  INSERT INTO R SFW;
  
  ```

- **Example:**
  
  ```sql
  INSERT INTO passengers VALUES ('David', NULL, 'Chiu', '888-88-8888');
  
  INSERT INTO passengers SELECT * FROM Student; -- now all students are passengers!
  ```
SQL: Deleting Tuples

- Relational Algebra Syntax: \( R = R - E \)

- Example:

\[
player \leftarrow player - \sigma_{\text{Salary}>60000}(player)
\]

- SQL Syntax:

```
DELETE FROM R WHERE C;
```

-- The WHERE clause is optional!!!
-- C is assumed true if not given.

- Example:

```
DELETE FROM player WHERE Salary > 60000;
```

```
DELETE FROM player; -- OMG what did I just do??
```
SQL: Updating Tuples

- Relational Algebra Syntax: \( R \leftarrow \Pi_{E_1, \ldots, E_k}(R) \)

- SQL Syntax:

  ```sql
  UPDATE R SET a1 = e1, ..., ak = ek WHERE C;
  -- e_i are SQL expressions
  -- C is assumed true if not given
  ```

- Example:

  ```sql
  UPDATE passengers SET m_name = 'John' WHERE ssn='888-88-8888';
  UPDATE players SET salary=(1.04 * salary);
  ```
Select-From-Where (SFW) Statements

- Recall this common R.A. expression: $\Pi_{a_1, \ldots, a_k}(\sigma_C(R))$

- In SQL, this projection-selection takes on a very common form:
  - Often referred to as an **SFW-query**

```sql
SELECT DISTINCT a1, a2, ..., a_k
FROM R
WHERE C;

-- Note 0: Anything in grey is optional
-- Note 1: Condition C is assumed true, if not given
-- Note 2: You can use '*' project all attributes
```
Examples of Simple SFW Statements

- **Contents of R**

<table>
<thead>
<tr>
<th>a</th>
<th>b</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>1</td>
<td>2</td>
</tr>
</tbody>
</table>

- **Definition of R**

  ```sql
  -- define the table
  create table R(
    a INTEGER,
    b INTEGER);

  -- insert some tuples
  insert into R values (1,2);
  insert into R values (1,5);
  insert into R values (1,2);
  ```

- **What are the results of:**
  - select a from R;
  - select distinct a from R;
  - select distinct a,b from R;
  - select * from R where b > 2;
The WHERE Clause

- Common comparison operators for the **WHERE** clause:
  - Recall that anything grey is optional

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>WHERE expr1 &lt; expr2</code></td>
<td>Checks if <code>expr1</code> is less than <code>expr2</code></td>
</tr>
<tr>
<td><code>WHERE expr1 &gt; expr2</code></td>
<td>Checks if <code>expr1</code> is greater than <code>expr2</code></td>
</tr>
<tr>
<td><code>WHERE expr1 &lt;= expr2</code></td>
<td>Checks if <code>expr1</code> is less than or equal to <code>expr2</code></td>
</tr>
<tr>
<td><code>WHERE expr1 &gt;= expr2</code></td>
<td>Checks if <code>expr1</code> is greater than or equal to <code>expr2</code></td>
</tr>
<tr>
<td><code>WHERE expr1 = expr2</code></td>
<td>Checks if <code>expr1</code> is equal to <code>expr2</code></td>
</tr>
<tr>
<td><code>WHERE expr1 &lt;&gt; expr2</code></td>
<td>Checks if <code>expr1</code> is not equal to <code>expr2</code> (sqlite also supports <code>!=</code>)</td>
</tr>
<tr>
<td><code>WHERE expr1 IS NOT NULL</code></td>
<td>Checks if <code>expr1</code> is not null</td>
</tr>
<tr>
<td><code>WHERE expr1 NOT IN (SWF)</code></td>
<td>Checks if <code>expr1</code> is not a member of the SWF result (sqlite only)</td>
</tr>
<tr>
<td><code>WHERE expr1 NOT LIKE string_expr</code></td>
<td>Checks if <code>expr1</code> does not match the regular expression <code>string_expr</code> (sqlite only)</td>
</tr>
<tr>
<td><code>WHERE expr1 NOT GLOB regexp</code></td>
<td>Checks if <code>expr1</code> does not match the global regular expression <code>regexp</code> (sqlite only)</td>
</tr>
</tbody>
</table>
Let's Run Some SFW Queries

Let's Run Some SFW Queries

### passengers

<table>
<thead>
<tr>
<th>f_name</th>
<th>m_name</th>
<th>l_name</th>
<th>ssn</th>
</tr>
</thead>
<tbody>
<tr>
<td>Homer</td>
<td>J</td>
<td>Simpson</td>
<td>111-11-1111</td>
</tr>
<tr>
<td>Bart</td>
<td>H</td>
<td>Simpson</td>
<td>444-44-4444</td>
</tr>
<tr>
<td>Lisa</td>
<td>G</td>
<td>Simpson</td>
<td>222-22-2222</td>
</tr>
<tr>
<td>Frank</td>
<td></td>
<td>Lovejoy</td>
<td>555-55-5555</td>
</tr>
<tr>
<td>Robert</td>
<td>N</td>
<td>Quimby</td>
<td>666-66-6666</td>
</tr>
<tr>
<td>Ned</td>
<td>T</td>
<td>Flanders</td>
<td>777-77-7777</td>
</tr>
<tr>
<td>Frank</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

### onboard

<table>
<thead>
<tr>
<th>ssn</th>
<th>flight_no</th>
<th>seat</th>
</tr>
</thead>
<tbody>
<tr>
<td>555-55-5555</td>
<td>6</td>
<td>32F</td>
</tr>
<tr>
<td>111-11-1111</td>
<td>2</td>
<td>2B</td>
</tr>
<tr>
<td>222-22-2222</td>
<td>4</td>
<td>1F</td>
</tr>
<tr>
<td>777-77-7777</td>
<td>5</td>
<td>25A</td>
</tr>
<tr>
<td>333-33-3333</td>
<td>3</td>
<td>25A</td>
</tr>
<tr>
<td>666-66-6666</td>
<td>1</td>
<td>30C</td>
</tr>
<tr>
<td>444-44-4444</td>
<td>7</td>
<td>30C</td>
</tr>
<tr>
<td>777-77-7777</td>
<td>3</td>
<td>25A</td>
</tr>
<tr>
<td>555-55-5555</td>
<td>4</td>
<td>25A</td>
</tr>
</tbody>
</table>

### plane

<table>
<thead>
<tr>
<th>tail_no</th>
<th>make</th>
<th>model</th>
<th>capacity</th>
<th>mph</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Boeing</td>
<td>747</td>
<td>525</td>
<td>570</td>
</tr>
<tr>
<td>2</td>
<td>Boeing</td>
<td>747</td>
<td>525</td>
<td>570</td>
</tr>
<tr>
<td>3</td>
<td>Airbus</td>
<td>A350</td>
<td>270</td>
<td>580</td>
</tr>
<tr>
<td>4</td>
<td>McDonnel Douglas</td>
<td>DC10</td>
<td>380</td>
<td>610</td>
</tr>
<tr>
<td>5</td>
<td>Airbus</td>
<td>A380</td>
<td>200</td>
<td>500</td>
</tr>
</tbody>
</table>

### flight

<table>
<thead>
<tr>
<th>flight_no</th>
<th>dep_loc</th>
<th>dep_time</th>
<th>arr_loc</th>
<th>arr_time</th>
<th>tail_no</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Springfield, IL</td>
<td>7:15</td>
<td>Chicago, IL</td>
<td>7:45</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>Columbus, OH</td>
<td>16:00</td>
<td>Portland, OR</td>
<td>22:00</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>New York, NY</td>
<td>12:30</td>
<td>Miami, FL</td>
<td>13:00</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>Paris, France</td>
<td>15:00</td>
<td>Munich, Germany</td>
<td>17:40</td>
<td>1</td>
</tr>
<tr>
<td>5</td>
<td>Hartford, CT</td>
<td>9:00</td>
<td>Phoenix, AZ</td>
<td>12:00</td>
<td>1</td>
</tr>
<tr>
<td>6</td>
<td>Miami, FL</td>
<td>10:45</td>
<td>Austin, TX</td>
<td>13:30</td>
<td>2</td>
</tr>
<tr>
<td>7</td>
<td>Akron, OH</td>
<td>14:20</td>
<td>Hartford, CT</td>
<td>16:45</td>
<td>3</td>
</tr>
</tbody>
</table>

sqlite> .read airport-schema.sql  -- creates the tables and constraints
sqlite> .read airport-populate.sql -- populates the tables with above data

(Find these files on the course page!)
Alternative to SQLite CLI

- There are also lots of free SQLite GUIs available
  - DB Browser for SQLite
- I'll use DB Browser in my demos.
  - Let's create that airport DB again!
Topics

- Structured Query Language (SQL)
  - Data Definition Language (DDL)
  - Data Manipulation Language (DML)
    - Select
      - from, where
      - set operations
      - joins
      - order by
      - sub-queries
      - aggregation and grouping
    - Insert
    - Delete
    - Update
Set Operators: union, intersect, except

- Remember, to apply any of these operations, the two relations must be compatible.
  - UNION, INTERSECT, and EXCEPT uses set-semantics

SQL Syntax:

```
SFW1 UNION ALL SFW2
SFW1 INTERSECT SFW2
SFW1 EXCEPT SFW2
```

(ALL keyword retains duplicates - means to apply bag-semantics!)
Example with the [ALL] Keyword

- Return a set of all last names and plane models

```sql
sqlite> select l_name from passengers
union
select model from plane;
```

- When using `union all` instead:

```sql
sqlite> select l_name from passengers
union all
select model from plane;
```
Examples

- Return the SSNs of everyone not named Bart (using set difference)
  - Relational algebra first:
    \[
    \pi_{ssn}(\text{passengers}) \setminus \pi_{ssn}(\sigma_{fname='Bart'}(\text{passengers}))
    \]
  - SQL?
    ```sql
    SELECT ssn FROM passengers
    EXCEPT
    SELECT ssn FROM passengers WHERE fname='Bart';
    ```
Cartesian Product

- The cartesian product: $R_1 \times R_2$
- SQL syntax: 

  ```sql
  SELECT * FROM R1, R2, R3, R4, ...;
  ```

  - Example:

    ```sql
    sqlite> select *
           from R,S;
    a1|b1|e1|f1|g1
    a1|b1|e2|f2|g2
    a2|b2|e1|f1|g1
    a2|b2|e2|f2|g2
    a3|b3|e1|f1|g1
    a3|b3|e2|f2|g2
    sqlite>
    ```
Let's Do a Natural Join the Old Way

- Return all seat numbers for the Simpsons in this format:
  - (f_name, l_name, seat)
  - Relational algebra first.... then SQL

<table>
<thead>
<tr>
<th>passengers</th>
<th>onboard</th>
</tr>
</thead>
<tbody>
<tr>
<td>f_name</td>
<td>m_name</td>
</tr>
<tr>
<td>Homer</td>
<td>J</td>
</tr>
<tr>
<td>Bart</td>
<td>H</td>
</tr>
<tr>
<td>Lisa</td>
<td>G</td>
</tr>
<tr>
<td>Frank</td>
<td></td>
</tr>
<tr>
<td>Robert</td>
<td>N</td>
</tr>
<tr>
<td>Ned</td>
<td>T</td>
</tr>
<tr>
<td>Frank</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Topics

- Structured Query Language (SQL)
  - Data Definition Language (DDL)
  - Data Manipulation Language (DML)
    - Select
      - from, where
      - set operations
      - joins
    - order by
    - sub-queries
    - aggregation and grouping
    - Insert
    - Delete
    - Update
Theta Join

- The $\theta$-join is defined: $\sigma_\theta(R_1 \times R_2) = R_1 \bowtie_\theta R_2$

- There's no keyword in SQL for theta-join, but we know:

  \[
  R_1 \times R_2 \text{ is the same as } \text{ FROM } R_1, R_2
  \]

- The rest? Just the **WHERE** clause to formulate the Join condition

  \[
  \text{SELECT } * \text{ FROM } R_1, R_2 \text{ WHERE } \theta;
  \]