Outline

• The Project
• Nulls (6.1.6)
• Outer joins (6.3.8)
• Database Modifications (6.5)
NULLS in SQL

• Whenever we don’t have a value, we can put a NULL
• Can mean many things:
  – Value does not exist
  – Value exists but is unknown
  – Value not applicable
  – Etc.
• The schema specifies for each attribute if can be null *(nullable attribute)* or not
• How does SQL cope with tables that have NULLs?
Null Values

• If \( x = \text{NULL} \) then \( 4 \times (3 - x) / 7 \) is still \( \text{NULL} \)

• If \( x = \text{NULL} \) then \( x = \text{“Joe”} \) is \( \text{UNKNOWN} \)

• In SQL there are three boolean values:

  \[
  \begin{align*}
  \text{FALSE} & = 0 \\
  \text{UNKNOWN} & = 0.5 \\
  \text{TRUE} & = 1
  \end{align*}
  \]
Null Values

- \( C_1 \text{ AND } C_2 = \min(C_1, C_2) \)
- \( C_1 \text{ OR } C_2 = \max(C_1, C_2) \)
- \( \text{NOT } C_1 = 1 - C_1 \)

**Example:**

```sql
SELECT * FROM Person WHERE (age < 25) AND (height > 6 OR weight > 190)
```

Rule in SQL: include only tuples that yield TRUE
Null Values

Unexpected behavior:

```sql
SELECT *
FROM Person
WHERE age < 25 OR age >= 25
```

Some Persons are not included!
Null Values

Can test for NULL explicitly:

- x IS NULL
- x IS NOT NULL

```
SELECT * 
FROM    Person 
WHERE   age < 25  OR  age >= 25 OR age IS NULL
```

Now it includes all Persons
Outerjoins

Explicit joins in SQL = “inner joins”:
Product(name, category)
Purchase(prodName, store)

```
SELECT Product.name, Purchase.store
FROM   Product JOIN Purchase ON
       Product.name = Purchase.prodName
```

Same as:
```
SELECT Product.name, Purchase.store
FROM   Product, Purchase
WHERE  Product.name = Purchase.prodName
```

But Products that never sold will be lost!
Outerjoins

Left outer joins in SQL:

Product(name, category)
Purchase(prodName, store)

```
SELECT Product.name, Purchase.store
FROM Product LEFT OUTER JOIN Purchase
ON Product.name = Purchase.prodName
```
<table>
<thead>
<tr>
<th>Name</th>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>gadget</td>
</tr>
<tr>
<td>Camera</td>
<td>Photo</td>
</tr>
<tr>
<td>OneClick</td>
<td>Photo</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>ProdName</th>
<th>Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Wiz</td>
</tr>
<tr>
<td>Camera</td>
<td>Ritz</td>
</tr>
<tr>
<td>Camera</td>
<td>Wiz</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Name</th>
<th>Store</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>Wiz</td>
</tr>
<tr>
<td>Camera</td>
<td>Ritz</td>
</tr>
<tr>
<td>Camera</td>
<td>Wiz</td>
</tr>
<tr>
<td>OneClick</td>
<td>NULL</td>
</tr>
</tbody>
</table>
Application

Compute, for each product, the total number of sales in ‘September’

Product(name, category)
Purchase(prodName, month, store)

```
SELECT Product.name, count(*)
FROM    Product, Purchase
WHERE   Product.name = Purchase.prodName
        and  Purchase.month = 'September'
GROUP BY Product.name
```

What’s wrong?
Application

Compute, for each product, the total number of sales in ‘September’

Product(name, category)
Purchase(prodName, month, store)

```sql
SELECT Product.name, count(*)
FROM   Product
LEFT OUTER JOIN Purchase
        ON Product.name = Purchase.prodName
        and  Purchase.month = 'September'
GROUP BY Product.name
```

Now we also get the products who sold in 0 quantity
Outer Joins

• Left outer join:
  – Include the left tuple even if there’s no match

• Right outer join:
  – Include the right tuple even if there’s no match

• Full outer join:
  – Include the both left and right tuples even if there’s no match
Modifying the Database

Three kinds of modifications

• Insertions
• Deletions
• Updates

Sometimes they are all called “updates”
Insertions

General form:

\[
\text{INSERT INTO } R(\text{A1},\ldots, \text{An}) \ \text{VALUES} \ (v1,\ldots, vn)
\]

Example: Insert a new purchase to the database:

\[
\text{INSERT INTO } \text{Purchase(}\text{buyer, seller, product, store)} \ \\
\text{VALUES} \ (\text{Joe'\text{', 'Fred', 'wakeup-clock-espresso-machine', 'The Sharper Image'})
\]

Missing attribute $\rightarrow$ NULL.
May drop attribute names if give them in order.
Insertions

```
INSERT INTO PRODUCT(name)

SELECT DISTINCT Purchase.product
FROM Purchase
WHERE Purchase.date > "10/26/01"
```

The query replaces the VALUES keyword. Here we insert many tuples into PRODUCT
Insertion: an Example

**Product**

Product(name, listPrice, category)

**Purchase**

Purchase(prodName, buyerName, price)

**prodName** is foreign key in Product.name

Suppose database got corrupted and we need to fix it:

**Product**

<table>
<thead>
<tr>
<th>name</th>
<th>listPrice</th>
<th>category</th>
</tr>
</thead>
<tbody>
<tr>
<td>gizmo</td>
<td>100</td>
<td>gadgets</td>
</tr>
</tbody>
</table>

**Purchase**

<table>
<thead>
<tr>
<th>prodName</th>
<th>buyerName</th>
<th>price</th>
</tr>
</thead>
<tbody>
<tr>
<td>camera</td>
<td>John</td>
<td>200</td>
</tr>
<tr>
<td>gizmo</td>
<td>Smith</td>
<td>80</td>
</tr>
<tr>
<td>camera</td>
<td>Smith</td>
<td>225</td>
</tr>
</tbody>
</table>

Task: insert in **Product** all **prodNames** from **Purchase**
Insertion: an Example

```
INSERT INTO Product(name)
SELECT DISTINCT prodName
FROM Purchase
WHERE prodName NOT IN (SELECT name FROM Product)
```

<table>
<thead>
<tr>
<th>name</th>
<th>listPrice</th>
<th>category</th>
</tr>
</thead>
<tbody>
<tr>
<td>gizmo</td>
<td>100</td>
<td>Gadgets</td>
</tr>
<tr>
<td>camera</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>
Insertion: an Example

```
INSERT INTO Product(name, listPrice)
SELECT  DISTINCT prodName, price
FROM Purchase
WHERE prodName NOT IN (SELECT name FROM Product)
```

<table>
<thead>
<tr>
<th>name</th>
<th>listPrice</th>
<th>category</th>
</tr>
</thead>
<tbody>
<tr>
<td>gizmo</td>
<td>100</td>
<td>Gadgets</td>
</tr>
<tr>
<td>camera</td>
<td>200</td>
<td>-</td>
</tr>
<tr>
<td>camera ??</td>
<td>225 ??</td>
<td>-</td>
</tr>
</tbody>
</table>
Deletions

Example:

```
DELETE FROM PURCHASE
WHERE seller = 'Joe' AND
product = 'Brooklyn Bridge'
```

Factoid about SQL: there is no way to delete only a single occurrence of a tuple that appears twice in a relation.
Updates

Example:

```sql
UPDATE PRODUCT
SET price = price/2
WHERE Product.name IN
    (SELECT product
     FROM Purchase
     WHERE Date = 'Oct, 25, 1999');
```
Data Definition in SQL

So far we have seen the Data Manipulation Language, DML. Next: Data Definition Language (DDL)

Data types:

- Defines the types.

Data definition: defining the schema.

- Create tables
- Delete tables
- Modify table schema

Indexes: to improve performance
Creating Tables

CREATE TABLE Person(
    name VARCHAR(30),
    social-security-number INT,
    age INT,
    city VARCHAR(30),
    gender BIT(1),
    Birthdate DATE
);
Deleting or Modifying a Table

Deleting:

Example: \textbf{DROP} Person; \hspace{1cm} Exercise with care !!

Altering: (adding or removing an attribute).

Example:

\begin{verbatim}
ALTER TABLE Person
ADD phone CHAR(16);

ALTER TABLE Person
DROP age;
\end{verbatim}

What happens when you make changes to the schema?
Default Values

Specifying default values:

```sql
CREATE TABLE Person(
    name          VARCHAR(30),
    social-security-number  INT,
    age           SHORTINT   DEFAULT 100,
    city      VARCHAR(30) DEFAULT 'Seattle',
    gender          CHAR(1)  DEFAULT '?',
    Birthdate                         DATE
)
```

The default of defaults: NULL
Indexes

**REALLY** important to speed up query processing time.

Suppose we have a relation

Person (name, age, city)

```sql
SELECT *
FROM Person
WHERE name = "Smith"
```

Sequential scan of the file Person may take long
Indexes

• Create an index on name:

B+ trees have fan-out of 100s: max 4 levels!
Will discuss in the second half of this course
Creating Indexes

Syntax:

```
CREATE INDEX nameIndex ON Person(name)
```
Creating Indexes

Indexes can be useful in range queries too:

CREATE INDEX ageIndex ON Person (age)

B+ trees help in:

SELECT * FROM Person WHERE age > 25 AND age < 28

Why not create indexes on everything?
Creating Indexes

Indexes can be created on more than one attribute:

Example:

```
CREATE INDEX doubleindex ON Person (age, city)
```

Helps in:

```
SELECT * FROM Person WHERE age = 55 AND city = “Seattle”
```

and even in:

```
SELECT * FROM Person WHERE age = 55
```

But not in:

```
SELECT * FROM Person WHERE city = “Seattle”
```
The Index Selection Problem

• Why not build an index on every attribute? On every pair of attributes? Etc.?

• The index selection problem is hard: balance the query cost v.s. the update cost, in a large application workload