Three Mechanisms for Storage:
SharedPreferences, Firebase, SQL
To Do for Sunday Night @ Midnight

- App Description
- Write Use Cases and Database Schema for your App
App Requirements

- Interactivity between users
- Multiple screens
- Use of one Android feature not taught in this class: e.g.,
  - GPS, accelerometer, gyroscope, or camera
  - Authentication (using Firebase)
  - Notifications (using Firebase)
  - User Data (e.g., phonebook, calendar)
  - Canvas (for custom widgets / graphics)
Structured Query Language (SQL)

- Standard language for interfacing with databases

- SQL functions fit into 2 broad categories:
  - Data definition language
    - Create database objects, such as tables, indexes, and views
      - Define access rights to those database objects
  - Data manipulation language
    - Includes commands to insert, update, delete, and retrieve data within database tables
SQL

- SQL is relatively easy to learn
- Basic command set has vocabulary of less than 100 words
- Nonprocedural language
- American National Standards Institute (ANSI) prescribes a standard SQL
  - But, several SQL dialects exist
# Tables in SQL

## Product

<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
<th>Category</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Photography</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Household</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>

**Table name**: Product

**Attribute names**: PName, Price, Category, Manufacturer

**Tuples or rows**
Tables Explained

- The *schema* of a table is the table name and its attributes:
  \[ \text{Product(PName, Price, Category, Manufacturer)} \]

- *A key* is an attribute whose values are unique; we underline a key

\[ \text{Product(PName, Price, Category, Manufacturer)} \]
Data Types in SQL

- **Atomic types:**
  - Characters: CHAR(20), VARCHAR(50)
  - Numbers: INT, BIGINT, SMALLINT, FLOAT
  - Others: MONEY, DATETIME, ...

- Every attribute must have a primitive type
  - Hence tables are flat
Tables Explained

- A tuple = a record = a row

- A table = a set of tuples
  - Like a list...
  - ...but it is unordered:
    no `first()`, no `next()`, no `last()`.
SQL Query

Basic form: (plus many many more bells and whistles)

```
SELECT <attributes>
FROM <one or more relations>
WHERE <conditions>
```
### Simple SQL Query

**Product**

<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
<th>Category</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Photography</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Household</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>

**SQL Query**

```
SELECT * 
FROM Product 
WHERE category= 'Gadgets'
```

**"selection"**

<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
<th>Category</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
</tbody>
</table>
Simple SQL Query

```
SELECT PName, Price, Manufacturer
FROM Product
WHERE Price > 100
```

<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
<th>Category</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Photography</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Household</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>

“selection” and “projection”
Notation

Product(PName, Price, Category, Manufacturer)

SELECT PName, Price, Manufacturer
FROM Product
WHERE Price > 100

Answer(PName, Price, Manufacturer)
Details

- **Case insensitive:**
  - Same: SELECT  Select  select
  - Same: Product  product
  - Different: ‘Seattle’ ‘seattle’

- **Constants:**
  - ‘abc’ - yes
  - “abc” - no
The LIKE operator

```
SELECT * 
FROM Products
WHERE PName LIKE '%gizmo%'
```

- **s LIKE p**: pattern matching on strings
- **p** may contain two special symbols:
  - `%` = any sequence of characters
  - `_` = any single character
Eliminating Duplicates

**SELECT**  **DISTINCT** category  
**FROM**  Product

Compare to:

**SELECT**  category  
**FROM**  Product

<table>
<thead>
<tr>
<th>Category</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gadgets</td>
</tr>
<tr>
<td>Photography</td>
</tr>
<tr>
<td>Household</td>
</tr>
</tbody>
</table>
Ordering the Results

SELECT  pname, price, manufacturer
FROM    Product
WHERE   category= 'gizmo' AND price > 50
ORDER BY price, pname

Ties are broken by the second attribute on the ORDER BY list, etc.

Ordering is ascending, unless you specify the DESC keyword.
<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
<th>Category</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Photography</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Household</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>

**SELECT DISTINCT** category
FROM Product
ORDER BY category

**SELECT** Category
FROM Product
ORDER BY PName

**SELECT DISTINCT** category
FROM Product
ORDER BY PName

Category: gadgets, household, photography

Gadgets
Household
Gadgets
Photography

Non-deterministic?
# Keys and Foreign Keys

## Company

<table>
<thead>
<tr>
<th>CName</th>
<th>StockPrice</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>GizmoWorks</td>
<td>25</td>
<td>USA</td>
</tr>
<tr>
<td>Canon</td>
<td>65</td>
<td>Japan</td>
</tr>
<tr>
<td>Hitachi</td>
<td>15</td>
<td>Japan</td>
</tr>
</tbody>
</table>

## Product

<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
<th>Category</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Photography</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Household</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>
Joins

Product (pname, price, category, manufacturer)
Company (cname, stockPrice, country)

Find all products under $200 manufactured in Japan; return their names and prices.

```
SELECT PName, Price
FROM Product, Company
WHERE Manufacturer=CName AND Country='Japan'
AND Price <= 200
```
### Joins

#### Product

<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
<th>Category</th>
<th>Manufacturer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Gizmo</td>
<td>$19.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>Powergizmo</td>
<td>$29.99</td>
<td>Gadgets</td>
<td>GizmoWorks</td>
</tr>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
<td>Photography</td>
<td>Canon</td>
</tr>
<tr>
<td>MultiTouch</td>
<td>$203.99</td>
<td>Household</td>
<td>Hitachi</td>
</tr>
</tbody>
</table>

#### Company

<table>
<thead>
<tr>
<th>Cname</th>
<th>StockPrice</th>
<th>Country</th>
</tr>
</thead>
<tbody>
<tr>
<td>GizmoWorks</td>
<td>25</td>
<td>USA</td>
</tr>
<tr>
<td>Canon</td>
<td>65</td>
<td>Japan</td>
</tr>
<tr>
<td>Hitachi</td>
<td>15</td>
<td>Japan</td>
</tr>
</tbody>
</table>

**SELECT** PName, Price
**FROM** Product, Company
**WHERE** Manufacturer=CName AND Country= 'Japan' AND Price <= 200

<table>
<thead>
<tr>
<th>PName</th>
<th>Price</th>
</tr>
</thead>
<tbody>
<tr>
<td>SingleTouch</td>
<td>$149.99</td>
</tr>
</tbody>
</table>

---
Tuple Variables

Person\((p_{\text{name}}, \text{address}, \text{worksfor})\)
Company\((c_{\text{name}}, \text{address})\)

```
SELECT DISTINCT \text{pname, address}
FROM \text{Person, Company}
WHERE \text{worksfor = cname}
```

Which address?

```
SELECT DISTINCT \text{Person.p\_name, Company.address}
FROM \text{Person, Company}
WHERE \text{Person.worksfor = Company.c\_name}
```

```
SELECT DISTINCT x.p\_name, y.address
FROM \text{Person AS x, Company AS y}
WHERE x.worksfor = y.c\_name
```
Aggregation

```sql
SELECT avg(price)
FROM Product
WHERE maker="Toyota"
```

```sql
SELECT count(*)
FROM Product
WHERE year > 1995
```

SQL supports several aggregation operations:

- sum, count, min, max, avg

Except count, all aggregations apply to a single attribute
Aggregation: Count

COUNT applies to duplicates, unless otherwise stated:

```sql
SELECT Count(category) 
FROM Product 
WHERE year > 1995
```

same as `Count(*)`

We probably want:

```sql
SELECT Count(DISTINCT category) 
FROM Product 
WHERE year > 1995
```
Modifying the Database

Three kinds of modifications

- Insertions
- Deletions
- Updates

Sometimes they are all called “updates”
Insertions

General form:

```
INSERT INTO R(A1, ..., An) VALUES (v1, ..., vn)
```

Example: Insert a new purchase to the database:

```
INSERT INTO Purchase(buyer, seller, product, store)
VALUES ( 'Joe', 'Fred', 'wakeup-clock-espresso-machine', 'The Sharper Image' )
```

Missing attribute → 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(name, listPrice, category)
Purchase(prodName, buyerName, price)

prodName is foreign key in Product.name

Suppose database got corrupted and we need to fix it:

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>

*Depends on the implementation*
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:

```
UPDATE PRODUCT
SET price = price/2
WHERE Product.name IN
    (SELECT product
     FROM Purchase
     WHERE Date = ‘Oct, 25, 1999’);
```