MySQL
CS 360 Internet Programming

Daniel Zappala

Brigham Young University
Computer Science Department
Starting the Command Interpreter

```
1  % mysql -h ilab.cs.byu.edu -uname -p
2
3  > source statements.sql
```

- will prompt for password
Creating and Using Databases

1  > `CREATE DATABASE juicestore;`
2
3  > `use juicestore;`
Creating Tables

```sql
1 CREATE TABLE customer ( // name the table
2   cust_id int(5) NOT NULL, // specify attributes
3   surname varchar(50),
4   firstname varchar(50),
5   initial char(1),
6   title_id int(3),
7   address varchar(50),
8   city varchar(50),
9   state varchar(20),
10  zipcode varchar(10),
11  country_id int(4),
12  phone varchar(15),
13  birth_date char(10),
14  PRIMARY KEY (cust_id) // create the primary key
15  ) type=MyISAM; // does not support transactions
```
Attribute Types

- `int(length)`: integer with a maximum length
- `decimal(width[,decimal_digits])`: float
- `datetime`: date and time in the format YYYY-MM-DD HH:MM:SS
- `time`: time in the format HH:MM:SS
- `date`: date in the format YYYY-MM-DD.
- `timestamp`: date and time in the format YYYYMMDDHHMMSS.
  - first-occurring timestamp attribute in a row is set to the current date and time when that created or modified.
  - timestamp will also be updated if you set it to NULL
- `varchar(length)`: unpadded, variable-length string
- `char(length)`: padded, fixed-length string
- `blob`: stores up to 64 KB of data
Attribute Modifiers

- **NOT NULL**: attribute must have a value
- **DEFAULT**: default value
- **zerofill**: left-pads a number with zeros
- **unsigned**: only positive values, doubles the maximum positive value
- **auto_increment**: automatically increments to next integer when set to NULL
Keys

- **primary key**: uniquely identifies a record
- can add additional keys
  - database will create an index for each key to provide faster lookups based on the key
  - each index takes additional space and must be updated for each insert, delete, modify operation
Deleting Databases and Tables

1. DROP TABLE customer;
2. 
3. DROP DATABASE juicestore;
4. 
5. DROP DATABASE IF EXISTS juicestore;
6. 
7. DROP TABLE IF EXISTS customer;
Inserting Data

1   INSERT INTO customer VALUES (1, 'Williams', 'Lucy', 'E', 3,
2   '272 Station St', 'Carlton North', 'VIC', '3054', 12, '(613)83008460',
3   '2002-07-02');

- number of values inserted must match the number of attributes
- must know ordering of attributes in table: use SHOW COLUMNS FROM customer
- may include NULL if the attribute allows this value
- may insert multiple rows at a time
Inserting Data

1. `INSERT INTO customer SET cust_id = 1, surname = 'Williams',
   firstname = 'Lucy', initial='E', title_id=3,
   address='272 Station St', city='Carlton North',
   state='VIC', zipcode='3054', country_id=12,
   phone='(613)83008460', birth_date='2002-07-10';

- list attribute names explicitly
- may skip some attributes
- may use a different attribute order
Default Values and Auto-Increment

- **default values**
  - if attribute is not included in INSERT, it is set to DEFAULT value if specified
  - if no DEFAULT value and NOT_NULL is not set, the value is set to NULL
  - if no DEFAULT and NOT_NULL is set, then integers are set to 0, and strings to ""

- **auto increment**
  - insert NULL as value for an attribute with auto increment set
  - only one attribute in a table may have this feature
Deleting Data

1. `DELETE FROM customer;`
   - deletes all records in customer table

2. `DELETE FROM customer WHERE cust_id = 1;`

3. `DELETE FROM customer WHERE surname = 'Smith';`
   - deletes only matching records
Updating Data

1. `UPDATE customer SET state = upper(state);`
2. `UPDATE customer SET state = upper(state), city = upper(city);`

- updates all records in customer table

1. `UPDATE customer SET surname = 'Smith' WHERE cust_id = 7;`
2. `UPDATE customer SET zipcode = '3001' WHERE city = 'Melbourne';`
Basic Query

1. `SELECT surname, firstname FROM customer;

<table>
<thead>
<tr>
<th>surname</th>
<th>firstname</th>
</tr>
</thead>
<tbody>
<tr>
<td>Marzalla</td>
<td>Dimitria</td>
</tr>
<tr>
<td>LaTrobe</td>
<td>Anthony</td>
</tr>
<tr>
<td>Fong</td>
<td>Nicholas</td>
</tr>
<tr>
<td>Stribling</td>
<td>James</td>
</tr>
</tbody>
</table>

4 rows in set (0.04 sec)

14. `SELECT * FROM region;`
WHERE Clauses

1. `SELECT region_name FROM region WHERE region_id <= 3;

<table>
<thead>
<tr>
<th>region_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>All</td>
</tr>
<tr>
<td>Goulburn Valley</td>
</tr>
<tr>
<td>Rutherglen</td>
</tr>
</tbody>
</table>

9. `3 rows in set (0.01 sec)`
Complex WHERE Clauses

1. SELECT cust_id FROM customer
2. WHERE (surname='Marzalla' AND firstname LIKE 'M %') OR
3. birth_date='1980-07-14';

<table>
<thead>
<tr>
<th>cust_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>440</td>
</tr>
<tr>
<td>493</td>
</tr>
</tbody>
</table>

2 rows in set (0.01 sec)
### Sorting Output

1. **SELECT** surname, firstname, initial **FROM** customer
2. **WHERE** city = 'Coonawarra' **OR** city = 'Longwood'
3. **ORDER BY** surname, firstname, initial;

<table>
<thead>
<tr>
<th>surname</th>
<th>firstname</th>
<th>initial</th>
</tr>
</thead>
<tbody>
<tr>
<td>Archibald</td>
<td>Belinda</td>
<td>Q</td>
</tr>
<tr>
<td>Chester</td>
<td>Marie</td>
<td>S</td>
</tr>
<tr>
<td>Dalion</td>
<td>Marie</td>
<td>C</td>
</tr>
<tr>
<td>Eggelston</td>
<td>Martin</td>
<td>E</td>
</tr>
<tr>
<td>Florenini</td>
<td>Melinda</td>
<td>O</td>
</tr>
<tr>
<td>Holdenson</td>
<td>Jasmine</td>
<td>F</td>
</tr>
<tr>
<td>Mellaseca</td>
<td>Craig</td>
<td>Y</td>
</tr>
<tr>
<td>Mockridge</td>
<td>Dimitria</td>
<td>I</td>
</tr>
</tbody>
</table>
### Grouping Output

- group matching rows
- report number of rows in each group
- **COUNT()**, **SUM()**, **MAX()**, **MIN()**, **AVG()**

```sql
1  SELECT city, COUNT(*) FROM customer GROUP BY city;
```

<table>
<thead>
<tr>
<th>city</th>
<th>COUNT(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alexandra</td>
<td>14</td>
</tr>
<tr>
<td>Armidale</td>
<td>7</td>
</tr>
<tr>
<td>Athlone</td>
<td>9</td>
</tr>
<tr>
<td>Bauple</td>
<td>6</td>
</tr>
<tr>
<td>Belmont</td>
<td>11</td>
</tr>
<tr>
<td>Bentley</td>
<td>10</td>
</tr>
<tr>
<td>Beralta</td>
<td>9</td>
</tr>
<tr>
<td>Broadmeadows</td>
<td>11</td>
</tr>
</tbody>
</table>
Combining Clauses

```
1 SELECT city, surname, firstname, count(*) FROM customer
2   WHERE state = 'VIC'
3 GROUP BY surname, firstname HAVING count(*) >= 2
4 ORDER BY city;
```

<table>
<thead>
<tr>
<th>city</th>
<th>surname</th>
<th>firstname</th>
<th>count(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Broadmeadows</td>
<td>Mellaseca</td>
<td>Anthony</td>
<td>2</td>
</tr>
<tr>
<td>Eleker</td>
<td>Leramonth</td>
<td>Harry</td>
<td>2</td>
</tr>
<tr>
<td>Kalimna</td>
<td>Galti</td>
<td>Nicholas</td>
<td>2</td>
</tr>
<tr>
<td>Lucknow</td>
<td>Mellili</td>
<td>Derryn</td>
<td>2</td>
</tr>
<tr>
<td>McLaren</td>
<td>Chester</td>
<td>Betty</td>
<td>2</td>
</tr>
</tbody>
</table>

5 rows in set (0.00 sec)
Join Queries

- match rows from tables based on relationship
- example: which customers that live in Australia have placed orders

```
1 SELECT juicery_name, region_name FROM juicery, region
2 ORDER BY juicery_name, region_name;

+-----------------+-----------------+
| juicery_name    | region_name     |
+-----------------+-----------------+
| Anderson and Sons Premium Juices | All             |
| Anderson and Sons Premium Juices | Barossa Valley  |
| Anderson and Sons Premium Juices | Coonawarra      |
| Anderson and Sons Premium Juices | Goulburn Valley |
| Anderson and Sons Premium Juices | Lower Hunter Valley |
+-----------------+-----------------+
```

- displays all possible combinations of juiceries and regions
want to output `juicery_name` and `region_name` values by matching rows from the juicery and region tables

query below automatically matches `region_id` attributes

```
SELECT juicery_name, region_name FROM juicery NATURAL JOIN region
ORDER BY juicery_name;
```

<table>
<thead>
<tr>
<th>juicery_name</th>
<th>region_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Anderson and Sons Premium Juices</td>
<td>Coonawarra</td>
</tr>
<tr>
<td>Anderson and Sons Juices</td>
<td>Coonawarra</td>
</tr>
<tr>
<td>Anderson Brothers Group</td>
<td>Rutherglen</td>
</tr>
<tr>
<td>Anderson Creek Group</td>
<td>Riverland</td>
</tr>
<tr>
<td>Anderson Daze Group</td>
<td>Rutherglen</td>
</tr>
</tbody>
</table>
INNER Join

- finds the intersection between two tables
- can explicitly list the relationship or use INNER JOIN or use NATURAL JOIN

---

```sql
1  SELECT DISTINCT surname, firstname, customer.cust_id
2   FROM customer, orders
3  WHERE customer.cust_id = orders.cust_id;

5  SELECT DISTINCT surname, firstname, customer.cust_id
6   FROM customer
7  INNER JOIN orders USING (cust_id);

9  SELECT DISTINCT surname, firstname, customer.cust_id
10  FROM customer
11  NATURAL JOIN orders;
```
ON Clause

- use **ON** when attributes don’t have the same name
- use **WHERE** to limit the rows of the output using additional conditions

```
1 SELECT juice_type.juice_type
2    FROM juice INNER JOIN juice_type
3    ON juice.juice_type=juice_type.juice_type_id
4    WHERE juice.juice_id=100;

5

6 SELECT juice_id FROM orders INNER JOIN items
7    ON orders.order_id=items.order_id AND orders.cust_id=items.cust_id
8    WHERE orders.cust_id=20 AND orders.order_id=1;
```
LEFT and RIGHT Outer Join

- outputs all rows from *left* side of the join, supplying NULL when there is no match from the right side
- list all the countries and customers who live in that country:

```
1   SELECT country, surname, firstname, cust_id
2   FROM countries LEFT JOIN customer USING (country_id);
```

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Australia</td>
<td>Stribling</td>
</tr>
<tr>
<td>2</td>
<td>Australia</td>
<td>Skerry</td>
</tr>
<tr>
<td>3</td>
<td>Australia</td>
<td>Cassisi</td>
</tr>
<tr>
<td>4</td>
<td>Australia</td>
<td>Krennan</td>
</tr>
<tr>
<td>5</td>
<td>Australia</td>
<td>Woodburne</td>
</tr>
<tr>
<td>6</td>
<td>Austria</td>
<td>NULL</td>
</tr>
<tr>
<td>7</td>
<td>Azerbaijan</td>
<td>NULL</td>
</tr>
<tr>
<td>8</td>
<td>Bahamas</td>
<td>NULL</td>
</tr>
</tbody>
</table>

- RIGHT: outputs all rows from the *right* side of the join, supplying NULL when there is no match from the left side
More Fun with Outer Join

- find the customers who have never placed an order:

```
1 SELECT surname, firstname, orders.cust_id
2 FROM customer LEFT JOIN orders USING (cust_id)
3 WHERE orders.cust_id IS NULL;
```

<table>
<thead>
<tr>
<th>surname</th>
<th>firstname</th>
<th>cust_id</th>
</tr>
</thead>
<tbody>
<tr>
<td>Sorrenti</td>
<td>Caitlyn</td>
<td>NULL</td>
</tr>
<tr>
<td>Mockridge</td>
<td>Megan</td>
<td>NULL</td>
</tr>
<tr>
<td>Krennan</td>
<td>Samantha</td>
<td>NULL</td>
</tr>
<tr>
<td>Dimitria</td>
<td>Melissa</td>
<td>NULL</td>
</tr>
<tr>
<td>Oaton</td>
<td>Mark</td>
<td>NULL</td>
</tr>
<tr>
<td>Cassisi</td>
<td>Joshua</td>
<td>NULL</td>
</tr>
</tbody>
</table>
User Variables

- save the result of a calculation to use later
- names of customers who bought the most expensive juice:

```
1   SELECT @max_cost:=max(cost) FROM inventory;
2
3   SELECT customer.cust_id, surname, firstname
4       FROM customer INNER JOIN items USING (cust_id)
5       INNER JOIN inventory USING (juice_id)
6       WHERE cost = @max_cost;
7
8   +-----------------+-----------------+------------------+
9   | cust_id | surname | firstname |
10  +-----------------+-----------------+------------------+
11  | 32     | Archibald   | Joshua           |
12  | 33     | Galti       | Lynette          |
13  | 44     | Mellili     | Michelle         |
14  | 54     | Woodestock  | George           |
15  | 71     | Mellaseca   | Lynette          |
16  | ...    |             |                  |
```
**UNION Clause**

- combine the results of two or more queries
- list the three oldest and three newest customers:

```sql
(SELECT cust_id, surname, firstname
 FROM customer ORDER BY cust_id LIMIT 3)
UNION
(SELECT cust_id, surname, firstname
 FROM customer ORDER BY cust_id DESC LIMIT 3);
```

<table>
<thead>
<tr>
<th>cust_id</th>
<th>surname</th>
<th>firstname</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Rosenthal</td>
<td>Joshua</td>
</tr>
<tr>
<td>2</td>
<td>Serrong</td>
<td>Martin</td>
</tr>
<tr>
<td>3</td>
<td>Leramonth</td>
<td>Jacob</td>
</tr>
<tr>
<td>650</td>
<td>Woodburne</td>
<td>Lynette</td>
</tr>
<tr>
<td>649</td>
<td>Krennan</td>
<td>Jim</td>
</tr>
<tr>
<td>648</td>
<td>Cassisi</td>
<td>Betty</td>
</tr>
</tbody>
</table>
Aliases

- shorthand for a table name, to save some typing

```
1 SELECT * FROM inventory i, juice j
2 WHERE i.juice_id = 183 AND i.juice_id = j.juice_id;
```

- find two customers with the same surname:

```
1 SELECT c1.cust_id, c2.cust_id FROM customer c1, customer c2
2 WHERE c1.surname = c2.surname AND c1.cust_id != c2.cust_id;
```
bookmark table

- id
- url
- tag

select all bookmarks with both the “blog” and “baseball” tags:

1. SELECT DISTINCT b1.bookmark FROM bookmarks b1, bookmarks b2
2. WHERE b1.id != b2.id AND b1.tag = "blog" AND b2.tag = "baseball";
Attribute Aliases

1. `SELECT surname AS s, firstname AS f FROM customer`
2. `WHERE surname = "Krennan" ORDER BY s, f;`

<table>
<thead>
<tr>
<th>s</th>
<th>f</th>
</tr>
</thead>
<tbody>
<tr>
<td>Krennan</td>
<td>Andrew</td>
</tr>
<tr>
<td>Krennan</td>
<td>Betty</td>
</tr>
<tr>
<td>Krennan</td>
<td>Caitlyn</td>
</tr>
<tr>
<td>Krennan</td>
<td>Caitlyn</td>
</tr>
<tr>
<td>Krennan</td>
<td>Dimitria</td>
</tr>
</tbody>
</table>
Introduction

- useful when you need to combine several queries
- note, next two examples could use a compound WHERE clause

1 # name of juiceries in the Margaret River region
2 SELECT juicery_name FROM juicery WHERE region_id =
3   (SELECT region_id FROM region
4     WHERE region_name = "Margaret River");

6 # name of region that makes juice #17
7 SELECT region_name FROM region WHERE region_id =
8   (SELECT region_id FROM juicery WHERE juicery_id =
9    (SELECT juicery_id FROM juice WHERE juice_id = 17));
Needed Nested Queries

• find the customer who has made the single largest purchase of a juice

1. `SELECT DISTINCT customer.cust_id FROM customer`  
2. `INNER JOIN items USING (cust_id)`  
3. `WHERE price = (SELECT MAX(price) FROM items);`
IN Clause

1  # find bookmarks with blog and baseball tag
2    SELECT id FROM bookmarks
3    WHERE tag="blog" AND bookmark_id IN
4        (SELECT id FROM bookmarks WHERE tag="baseball");
5
6  # find juices purchased by customers who placed six or more orders
7    SELECT DISTINCT juice_id FROM items WHERE cust_id IN
8        (SELECT customer.cust_id FROM customer
9            INNER JOIN orders USING (cust_id)
10           GROUP BY cust_id HAVING count(order_id) >= 6);
EXISTS Clause

- print results from outer query only if inner query returns results
- select the regions that have at least 35 juiceries:

```sql
1  SELECT region_name FROM region WHERE EXISTS
2     (SELECT * FROM juicery WHERE region.region_id = juicery.region_id
3     GROUP BY region_id HAVING count(*) > 35);
```