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

```
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',
2      firstname = 'Lucy', initial='E', title_id=3,
3      address='272 Station St', city='Carlton North',
4      state='VIC', zipcode='3054', country_id=12,
5      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

```
1 DELETE FROM customer WHERE cust_id = 1;
2
3 DELETE FROM customer WHERE surname = 'Smith';
```

- deletes only matching records

Updating Data

```
1 UPDATE customer SET state = upper(state);
2
3 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
3 UPDATE customer SET zipcode = '3001' WHERE city = 'Melbourne';
```

Basic Query

```
1  SELECT surname, firstname FROM customer;
2
3  +-----------+-----------+
4  | surname   | firstname |
5  +-----------+-----------+
6  | Marzalla  | Dimitria  |
7  | LaTrobe   | Anthony   |
8  | Fong      | Nicholas  |
9  | Stribling | James     |
10 +-----------+-----------+
11 4 rows in set (0.04 sec)
12
13
14 SELECT * FROM region;
```

WHERE Clauses

```
1  SELECT region_name FROM region WHERE region_id <= 3;
2  +-----------------+
3  | region_name     |
4  +-----------------+
5  | All             |
6  | Goulburn Valley |
7  | Rutherglen      |
8  +-----------------+
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';
4
5  +---------+
6  | cust_id |
7  +---------+
8  |     440 |
9  |     493 |
10 +---------+
11 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;
4
5
6  +-----------+-----------+---------+
7  | surname   | firstname | initial |
8  +-----------+-----------+---------+
9  | Archibald | Belinda   | Q       |
10 | Chester   | Marie     | S       |
11 | Dalion    | Marie     | C       |
12 | Eggelston | Martin    | E       |
13 | Florenini | Melinda   | O       |
14 | Holdenson | Jasmine   | F       |
15 | Mellaseca | Craig     | Y       |
16 | Mockridge | Dimitria  | I       |
```

Grouping Ouptut

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

```
1  SELECT city, COUNT(*) FROM customer GROUP BY city;
2
3  +--------------+----------+
4  | city         | COUNT(*) |
5  +--------------+----------+
6  | Alexandra    |       14 |
7  | Armidale     |        7 |
8  | Athlone      |        9 |
9  | Bauple       |        6 |
10 | Belmont      |       11 |
11 | Bentley      |       10 |
12 | Berala       |        9 |
13 | Broadmeadows |       11 |
```

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;
5
6  +-------------+-----------+-----------+----------+
7  | city        | surname   | firstname | count(*) |
8  +-------------+-----------+-----------+----------+
9  | Broadmeadows | Mellaseca | Anthony   |        2 |
10 | Eleker      | Leramonth | Harry     |        2 |
11 | Kalimna     | Galti     | Nicholas  |        2 |
12 | Lucknow     | Mellili   | Derryn    |        2 |
13 | McLaren     | Chester   | Betty     |        2 |
14 +-------------+-----------+-----------+----------+
15 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;
3
4  +-----------------------------------+-----------------------+
5  | juicery_name                      | region_name           |
6  +-----------------------------------+-----------------------+
7  | Anderson and Sons Premium Juices  | All                   |
8  | Anderson and Sons Premium Juices  | Barossa Valley        |
9  | Anderson and Sons Premium Juices  | Coonawarra            |
10 | Anderson and Sons Premium Juices  | Goulburn Valley       |
11 | Anderson and Sons Premium Juices  | Lower Hunter Valley   |
```

- **displays all possible combinations of juiceries and regions**

Inner/Natural Joins

- 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

```
1  SELECT juicery_name, region_name FROM juicery NATURAL JOIN region
2    ORDER BY juicery_name;
3
4  +-----------------------------------+-----------------------+
5  | juicery_name                      | region_name           |
6  +-----------------------------------+-----------------------+
7  | Anderson and Sons Premium Juices  | Coonawarra            |
8  | Anderson and Sons Juices          | Coonawarra            |
9  | Anderson Brothers Group           | Rutherglen            |
10 | Anderson Creek Group              | Riverland             |
11 | Anderson Daze Group               | Rutherglen            |
```

INNER Join

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

```
1  SELECT DISTINCT surname, firstname, customer.cust_id
2    FROM customer, orders
3    WHERE customer.cust_id = orders.cust_id;
4
5  SELECT DISTINCT surname, firstname, customer.cust_id
6    FROM customer
7    INNER JOIN orders USING (cust_id);
8
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);
3
4  | Australia  | Stribling | Michelle |  646 |
5  | Australia  | Skerry    | Samantha |  647 |
6  | Australia  | Cassisi   | Betty    |  648 |
7  | Australia  | Krennan   | Jim      |  649 |
8  | Australia  | Woodburne | Lynette  |  650 |
9  | Austria    | NULL      | NULL     | NULL |
10 | Azerbaijan | NULL      | NULL     | NULL |
11 | Bahamas    | NULL      | NULL     | NULL |
```

- 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;
4
5  +-----------+-----------+----------+
6  | surname   | firstname | cust_id  |
7  +-----------+-----------+----------+
8  | Sorrenti  | Caitlyn   |     NULL |
9  | Mockridge | Megan     |     NULL |
10 | Krennan   | Samantha  |     NULL |
11 | Dimitria  | Melissa   |     NULL |
12 | Oaton     | Mark      |     NULL |
13 | Cassisi   | Joshua    |     NULL |
```

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:

```
1  (SELECT cust_id, surname, firstname
2    FROM customer ORDER BY cust_id LIMIT 3)
3  UNION
4  (SELECT cust_id, surname, firstname
5    FROM customer ORDER BY cust_id DESC LIMIT 3);
6
7  +---------+-----------+-----------+
8  | cust_id | surname   | firstname |
9  +---------+-----------+-----------+
10 |       1 | Rosenthal | Joshua    |
11 |       2 | Serrong   | Martin    |
12 |       3 | Leramonth | Jacob     |
13 |     650 | Woodburne | Lynette   |
14 |     649 | Krennan   | Jim       |
15 |     648 | Cassisi   | Betty     |
16 +---------+-----------+-----------+
```

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;
```

More Fun with Aliases

- 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;
3
4  +----------+-----------+
5  | s        | f         |
6  +----------+-----------+
7  | Krennan  | Andrew    |
8  | Krennan  | Betty     |
9  | Krennan  | Caitlyn   |
10 | Krennan  | Caitlyn   |
11 | Krennan  | Dimitria  |
```

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");
5
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:

```
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);
```