Mysql accounts are now available on pe07

- As with Oracle, username = password = UNIX username
- Special for mysql: you have a database named <username>db
  - Access it from login on pe07.cs.umb.edu
  - Use: `pe07 mysql -u username -D usernamedb`
- To load our tables into your mysql account:
  - See README in tables directory:
    - Make file `.my.cnf` in your login directory as directed
    - Then use: `pe07 sh dbbook_mysql.sh` in your tables directory.
- Drop the database with:
  - `pe07 mysql -u username -D usernamedb < dropdb.sql`

**Structured Query Language**

- More on Constraints, Queries -

From Murach, Chap. 10:

**A table with column-level constraints**

```sql
CREATE TABLE vendors
(
  vendor_id     NUMBER       PRIMARY KEY,
  vendor_name   VARCHAR2(50) NOT NULL UNIQUE
)
```

**A table with named column-level constraints**

```sql
CREATE TABLE vendors
(
  vendor_id     NUMBER       PRIMARY KEY,
  vendor_name   VARCHAR2(50) NOT NULL UNIQUE
)
```

**A table with table-level constraints**

```sql
CREATE TABLE vendors
(
  vendor_id     NUMBER,
  vendor_name   VARCHAR2(50) NOT NULL,
  CONSTRAINT vendors_pk PRIMARY KEY (vendor_id),
  CONSTRAINT vendor_name_uq UNIQUE (vendor_name)
)
```

**A table with a two-column primary key constraint**

```sql
CREATE TABLE invoice_line_items
(
  invoice_id              NUMBER        NOT NULL,
  invoice_sequence        NUMBER        NOT NULL,
  line_item_description   VARCHAR2(100) NOT NULL,
  CONSTRAINT line_items_pk PRIMARY KEY (invoice_id, invoice_sequence)
)
```

**Terms to know**

- Constraint
- Column-level constraint
- Table-level constraint
- Not null constraint
- Unique constraint
- Primary key constraint
- Foreign key constraint

**The syntax of a column-level foreign key constraint**

```sql
[CONSTRAINT constraint_name]
REFERENCES table_name (column_name)
[ON DELETE {CASCADE|SET NULL}]
```

**The syntax of a table-level foreign key constraint**

```sql
[CONSTRAINT constraint_name]
FOREIGN KEY (column_name_1 [ , column_name_2]...)
REFERENCES table_name (column_name_1 [ , column_name_2]...)
[ON DELETE {CASCADE|SET NULL}]
```

"on delete cascade" is explained on pg. 71 of R&G.
A table with a column-level foreign key constraint

CREATE TABLE invoices
(
  invoice_id     NUMBER   PRIMARY KEY,
  vendor_id      NUMBER   REFERENCES vendors (vendor_id),
  invoice_number VARCHAR2(50) NOT NULL    UNIQUE,
  invoice_date   DATE
)

• Since vendor_id is the PK of vendors, we should be able to put simply “references vendors”, but listing the target column too works on more DBs, so best for portability
• mysql does not enforce column-level FKs, only table-level ones.
• Datatypes NUMBER and VARCHAR2 are Oracle-specific: better to use INT and VARCHAR

An INSERT statement that fails because a related row doesn’t exist

INSERT INTO invoices VALUES (1, 1, '1')

The response from the system

SQL Error: ORA-02291: integrity constraint (EX.INVOICES_FK_VENDORS) violated - parent key not found
*Cause: A foreign key value has no matching primary key value.
*Action: Delete the foreign key or add a matching primary key.

A constraint that uses the ON DELETE clause

CONSTRAINT invoices_fk_vendors
  FOREIGN KEY (vendor_id) REFERENCES vendors (vendor_id)
  ON DELETE CASCADE

• This means deletion of a vendor causes deletion of all its invoices.

The syntax for modifying the columns of a table

ALTER TABLE [schema_name.]table_name
{  
  ADD column_name data_type [column_attributes] |
  DROP COLUMN column_name |
  MODIFY column_name data_type [column_attributes] |
}
The syntax for modifying the constraints of a table

\[
\begin{align*}
&\text{ALTER TABLE table_name} \\
&\quad \{ \\
&\quad \quad \text{ADD} \quad \text{CONSTRAINT constraint_name constraint_definition \[DISABLE\]} \\
&\quad \quad \text{DROP} \quad \text{CONSTRAINT constraint_name} \\
&\quad \quad \text{ENABLE \[NOVALIDATE\]} \quad \text{constraint_name} \\
&\quad \quad \text{DISABLE} \quad \text{constraint_name} \\
&\}
\end{align*}
\]

A statement that adds a foreign key constraint

\[
\begin{align*}
&\text{ALTER TABLE invoices} \\
&\quad \text{ADD CONSTRAINT invoices_fk_vendors} \\
&\quad \text{FOREIGN KEY (vendor_id) \ REFERENCES vendors (vendor_id);} \\
\end{align*}
\]

A statement that adds a unique constraint

\[
\begin{align*}
&\text{ALTER TABLE vendors} \\
&\quad \text{ADD CONSTRAINT vendors_vendor_name_uq} \\
&\quad \text{UNIQUE (vendor_name);} \\
\end{align*}
\]

A statement that adds a not null constraint

\[
\begin{align*}
&\text{ALTER TABLE vendors} \\
&\quad \text{MODIFY vendor_name} \\
&\quad \text{CONSTRAINT vendors_vendor_name_nn NOT NULL;} \\
\end{align*}
\]

Example Schema

<table>
<thead>
<tr>
<th>Sailors</th>
<th>Boats</th>
</tr>
</thead>
<tbody>
<tr>
<td>sid</td>
<td>name</td>
</tr>
<tr>
<td>22</td>
<td>dustin</td>
</tr>
<tr>
<td>31</td>
<td>lubber</td>
</tr>
<tr>
<td>58</td>
<td>rusty</td>
</tr>
</tbody>
</table>

Reserves

<table>
<thead>
<tr>
<th>sid</th>
<th>bid</th>
<th>day</th>
</tr>
</thead>
<tbody>
<tr>
<td>22</td>
<td>101</td>
<td>10/10/96</td>
</tr>
<tr>
<td>31</td>
<td>103</td>
<td>11/12/96</td>
</tr>
<tr>
<td>58</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Query Example 1

• “Find names of sailors who’ve reserved boat #103”

SELECT ??

```
SELECT S.sname
FROM Sailors S, Reserves R
WHERE S.sid=R.sid AND R.bid=103
```

Query Example 2

• “Find names of sailors who’ve reserved a red boat”

SELECT ??

```
SELECT ??
```
Query Example 2

• “Find names of sailors who’ve reserved a red boat”

```
SELECT S.sname
FROM Sailors S, Reserves R, Boats B
    AND B.color='red';
```

Query Example 3

• “Find sailor ids who’ve reserved a red or a green boat; list each matching sailor id once”

```
SELECT DISTINCT S.sid
FROM Sailors S, Reserves R, Boats B
    AND (B.color='red' OR B.color='green');
```

```
SELECT DISTINCT R.sid
FROM Reserves R, Boats B
WHERE R.bid=B.bid
    AND (B.color='red' OR B.color='green');
```

Same query with set operations

```
SELECT S.sid
FROM Sailors S, Boats B, Reserves R
    AND B.color='red'
UNION
SELECT S.sid
FROM Sailors S, Boats B, Reserves R
    AND B.color='green'
  (can be simplified)
```

Query Example 4

• “Find sailor ids who’ve reserved a red and a green boat”

```
SELECT ??
```
Query Example 4 (can be simplified)

• “Find sailor ids who’ve reserved a red and a green boat”

```
SELECT S.sid
FROM Sailors S, Boats B1, Reserves R1,
     Boats B2, Reserves R2
WHERE S.sid=R1.sid AND R1.bid=B1.bid
AND S.sid=R2.sid AND R2.bid=B2.bid
AND (B1.color='red' AND B2.color='green')
```

```
Sailors | Boats
-------|-------
sid    | sname  | rating | age |
-------|--------|--------|-----|
bid    | name   | color  |
-------|--------|--------|
sid    | bid    | day    |
-------|--------|--------|
```

Same query with set operations

```
SELECT S.sid
FROM Sailors S, Boats B, Reserves R
AND B.color='red'
INTERSECT
SELECT S.sid
FROM Sailors S, Boats B, Reserves R
AND B.color='green'
```

Note: Some DBs don’t support INTERSECT, including mysql. All DBs support UNION.

A SELECT statement that retrieves all invoices between given dates

```
SELECT invoice_number, invoice_date, invoice_total
FROM invoices
WHERE invoice_date BETWEEN '01-MAY-2014' AND '31-MAY-2014'
ORDER BY invoice_date
```

How to concatenate string data

```
SELECT vendor_city, vendor_state,
       vendor_city || vendor_state 
FROM vendors
```

Note: Mysql doesn’t support ||: need to use concat(…)

How to format string data using literal values

```
SELECT vendor_name, 
    vendor_city || ', '  
    || vendor_state  
    || ' '  
    || vendor_zip_code  
    AS address 
FROM vendors
```

Terms to know

- String expression
- Literal value
- String literal (string constant)
- Concatenation operator
A SELECT statement that uses SUBSTR

```sql
SELECT vendor_contact_first_name, 
    vendor_contact_last_name, 
    SUBSTR(vendor_contact_first_name, 1, 1) || 
    SUBSTR(vendor_contact_last_name, 1, 1) AS initials 
FROM vendors
```

Substring function:
- Standard: SUBSTRING(str FROM start [FOR len])
- Mysql: standard or SUBSTR(…)
- Oracle: only SUBSTR(…)

A SELECT statement that uses TO_CHAR

```sql
SELECT 'Invoice: # ' || invoice_number 
    || ', dated ' || TO_CHAR(payment_date, 'MM/DD/YYYY') 
    || ' for $' || TO_CHAR(payment_total) 
    AS "Invoice Text" 
FROM invoices
```

TO_CHAR is Oracle-specific.

A SELECT statement that uses the MOD function

```sql
SELECT invoice_id, 
    MOD(invoice_id, 10) AS Remainder 
FROM invoices 
ORDER BY invoice_id
```

A SELECT statement that uses the Dual table

```sql
SELECT 'test' AS test_string, 10-7 AS test_calculation, SYSDATE AS test_date 
FROM Dual
```

- The Dual table is Oracle-specific.
- SYSDATE is Oracle-specific (holds current date/time, but default output is date only).
- Use CURRENT_TIMESTAMP for portability.

SQL Dialects and SQL functions

- A “SQL function” is a function that accepts values (strings, numbers, dates, etc.) and returns a value (no table processing here).
- Only a few SQL functions were standardized early enough (SQL-92) to have a chance of being uniformly supported in the products.
- SQL-92: char_length, substring, trim, current_time, current_timestamp (but none are “Entry-level”).
- How can we tell what SQL functions (and operators) are available for each product?
  - https://en.wikibooks.org/wiki/SQL_Dialects_Reference
  - We looked up mod, ||, to_char in class
  - Note the SQL standard (2011) column