CS 2451 Database Systems: Intro to SQL …

http://www.seas.gwu.edu/~bhagiweb/cs2541

Spring 2020

Instructor: Dr. Bhagi Narahari & R. Leontie

Next….SQL!

- Defining relational schema
  - Table definition
  - Specify constraints and keys
- Getting started with MySQL
- SQL Queries

Relational Model Definitions

- A ***relation*** is a table with columns and rows.
- An ***attribute*** is a named column of a relation.
  - A ***tuple*** is a row of a relation.
- A ***domain*** is a set of allowable values for one or more attributes.
- A ***relational database*** is a collection of normalized relations with distinct relation names.
- Key: set of attributes that uniquely identify a tuple/row
  - No two rows can have the same key value
- Primary key: one of the keys to the table
- Foreign key: if an attribute in one table is the primary key in another table
  - Provides “link” between tables

Recall: Schema Dessign & Relational Integrity

- Integrity rules are used to insure the data is accurate.
- ***Constraints*** are rules or restrictions that apply to the database and limit the data values it may store.
  - DBMS checks the constraints
- Types of constraints:
  - ***Domain constraint*** - Every value for an attribute must be an element of the attribute's domain or be `null`.

    `null` represents a value that is currently unknown or not applicable.

    `null` is not the same as zero or an empty string.
  - ***Entity integrity constraint*** - In a base relation, no attribute of a primary key can be null.
  - **Key constraint** – every relation must have a key; one of them chosen as primary key
  - ***Referential integrity constraint*** - If a foreign key exists in a relation, then the foreign key value must match a primary key value of a tuple in the referenced relation or be null.

Referential integrity and Foreign Keys

- Only students listed in the Students relation should be allowed to enroll for courses.
- Sid in Enrolled is foreign key referencing students
  - Sid is key for Students table

Enrolled

sid	cid	grade
53666	Jazz101	C
53666	Reggae203	B
53650	Topology112	A
53666	History105	B

Students

sid	name	login	age	gpa
53666	Jones	jones@cs	18	3.4
53688	Smith	smith@eecs	18	3.2
53650	Smith	smith@math	19	3.8

Next: SQL Module 1

- Specifying schema/table
- Specifying constraints in SQL

SQL: Structured Query Language

The standard language for relational data

- Invented by folks at IBM, esp. Don Chamberlin
- Actually not a great language…
- Beat a more elegant competing standard, QUEL, from Berkeley

Separated into a DML & DDL

SQL DML component based on relational algebra & calculus

- Data definition (DDL) – to define schema/tables
  - Define Schema
  - Define Constraints

SQL Basic Rules…read up on SQL syntax

- Some basic rules for SQL statements:
  - 1) There is a set of *reserved words* that cannot be used as names for database objects. (e.g. `SELECT`, `FROM`, `WHERE`)
  - 2) SQL is *case-insensitive*.
    Only exception is string constants. 'FRED' not the same as 'fred'.
  - 3) SQL is *free-format* and white-space is ignored.
  - 4) The semi-colon is often used as a statement terminator, although that is not always required.
  - 5) Date and time constants have defined format:
    Dates: 'YYYY-MM-DD' e.g. '1975-05-17'
    Times: 'hh:mm:ss[.f] ' e.g. '15:00:00'
    Timestamp: 'YYYY-MM-DD hh:mm:ss[.f] ' e.g. '1975-05-17 15:00:00'
  - 6) Two single quotes '' are used to represent a single quote character in a character constant. e.g. 'Master''s'.

SQL Query Language: DML

To query and retrieve data from the tables we have a:

- SELECT clause
  - What attributes you want
  - What relations/tables to search
  - What condition/predicate to apply

SQL and Relational Algebra

- The `SELECT` statement can be mapped directly to relational algebra.

- **SELECT** $A_1, A_2, \ldots, A_n$ */* this is projection*
- **FROM** $R_1, R_2, \ldots, R_m$ */* this is the cartesian prod*
- **WHERE** P */* this is selection op*

- is equivalent to:

$$\Pi_{A_1, A_2, \ldots, A_n}(\sigma_P(R_1 \times R_2 \times \ldots \times R_m))$$

More on this later…

SQL DDL

- SQL data definition language (DDL) allows users to:
  - add, modify, and drop tables
  - define and enforce integrity constraints
  - enforce security restrictions
  - Create views

SQL Identifiers and Data types…standard definitions you've seen before in other languages

- ***Identifiers*** are used to identify objects in the database such as tables, views, and columns.
  - The identifier is the name of the database object.
  - Rules for SQL identifiers…read notes
  - Note: Quoted or ***delimited identifiers*** enclosed in double quotes allow support for spaces and other characters. E.g. `"select"`
- Data types: each attribute has associated domain of values – i.e., each column has data type
  - The DBMS can perform implicit data type conversion when necessary
  - Can also do explicit conversion using CAST and CONVERT
- SQL also supports *user defined data types*
  - CREATE DOMAIN
  - Similar to typedef in C ?

SQL Data Types…similar to prog lang

Data Type	Description
BOOLEAN	TRUE or FALSE
CHAR	Fixed length string (padded with blanks) e.g. CHAR(10)
VARCHAR	Variable length string e.g. VARCHAR(50)
BIT	Bit string e.g. BIT(4) can store '0101'
NUMERIC or DECIMAL	Exact numeric data type e.g. NUMERIC(7,2) has a precision (max. digits) of 7 and scale of 2 (# of decimals) e.g. 12345.67
INTEGER	Integer data only
SMALLINT	Smaller space than INTEGER
FLOAT or REAL	Approximate numeric data types.
DOUBLE PRECISION	Precision dependent on implementation.
DATE	Stores YEAR, MONTH, DAY
TIME	Stores HOUR, MINUTE, SECOND
TIMESTAMP	Stores date and time data.
INTERVAL	Time interval.
CHARACTER LARGE OBJECT	Stores a character array (e.g. for a document)
BINARY LARGE OBJECT	Stores a binary array (e.g. for a picture, movie)

COMPANY Database Schema

EMPLOYEE

Fname	Minit	Lname	<u>Ssn</u>	Bdate	Address	Sex	Salary	Super_ssn	Dno

DEPARTMENT

Dname	<u>Dnumber</u>	Mgr_ssn	Mgr_start_date

DEPT_LOCATIONS

<u>Dnumber</u>	<u>Dlocation</u>

PROJECT

Pname	<u>Pnumber</u>	Plocation	Dnum

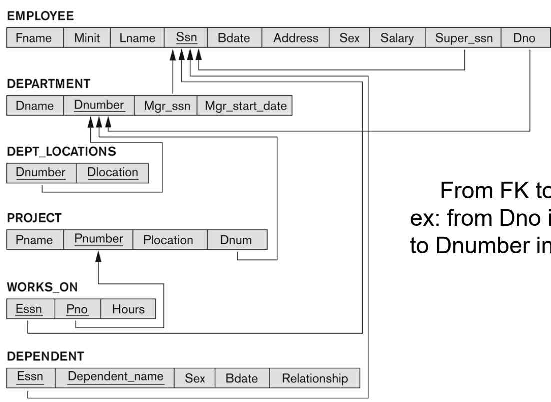
WORKS_ON

<u>Essn</u>	<u>Pno</u>	Hours

DEPENDENT

<u>Essn</u>	<u>Dependent_name</u>	Sex	Bdate	Relationship

Referential Integrity Constraints for COMPANY database



From FK to PK
ex: from Dno in EMP
to Dnumber in DEPT

Example Schema

- Relational database schema:

employee (ssn, fname, Minit, Lname,
bdate, address, Gender,salary,
superssn, Dno)
project (pnumber, pname, Plocation,
Dnum)
department (dnumber, dname, mgrssn,
Mgr_start_date)
workson (essn, pno, hours)

SQL CREATE TABLE

- The **CREATE TABLE** command is used to create a table in the database. A table consists of a table name, a set of fields with their names and data types, and specified constraints.

- The general form is:

```
CREATE TABLE tableName (
   attr1Name attr1Type  [attr1_constraints],
   attr2Name attr2Type  [attr2_constraints],
   ...
   attrMName attrMType  [attrM_constraints],
   [primary and foreign key constraints]
);
```

SQL CREATE TABLE Example

- The `CREATE TABLE` command for the `Emp` relation:

```
CREATE TABLE employee (
   ssn      CHAR(9),
   fname    VARCHAR(15) NOT NULL,
   minit    CHAR(1),
   lname    CHAR(15),
   bdate    DATE,
   sex      CHAR(1),
   salary   DECIMAL(10,2),
   superssn CHAR(9),
   dno      INT(4),
   );
```

SQL Constraints - Entity Integrity

- **Entity Integrity constraint** - The primary key of a table must contain a unique, non-null value for each row. The primary key is specified using the `PRIMARY KEY` clause.
  - e.g. `PRIMARY KEY (ssn)` (for Emp relation)
  - e.g. `PRIMARY KEY (essn,pno)` (for WorksOn relation)
  - It is also possible to use `PRIMARY KEY` right after defining the attribute in the `CREATE TABLE` statement.
- There can only be one primary key per relation, other candidate keys can be specified using `UNIQUE`:
  - e.g. **`UNIQUE (lname)`**

Another Example…'mini-banner'

- Create Students table
  - Info on students
- Takes table holds information about courses that students take.
  - Is sid same field in the two tables??

```
CREATE TABLE Students
    (sid: CHAR(20),
    name: CHAR(20),
    PRIMARY KEY (sid)) ;
```

```
CREATE TABLE Takes
    (sid: CHAR(20),
    cid: CHAR(20),
    grade: CHAR(2))
```

Specifying constraints on Takes table

- A reasonable condition/constraint:

"For a given student and course, there is a single grade"

```
CREATE TABLE Enrolled
        (sid: CHAR(20),
         cid: CHAR(20),
        grade: CHAR(2))
```

Does this schema have any problems ?

```
CREATE TABLE Enrolled2
        (sid CHAR(20)
         cid  CHAR(20),
        grade CHAR(2),
     PRIMARY KEY  (sid),
      UNIQUE (cid, grade) )
```

Effect of incorrect constraints….

- Enrolled1: "For a given student and course, there is a single grade." vs. Enrolled 2: "Students can take only one course, and receive a single grade for that course; further, no two students in a course receive the same grade."
- Used carelessly, an IC can prevent the storage of database instances that arise in practice!

```
CREATE TABLE Enrolled1
  (sid CHAR(20)
   cid  CHAR(20),
   grade CHAR(2),
   PRIMARY KEY  (sid,cid) )

CREATE TABLE Enrolled2
  (sid CHAR(20)
   cid  CHAR(20),
   grade CHAR(2),
   PRIMARY KEY  (sid),
   UNIQUE (cid, grade) )
```

SQL Constraints - Referential Integrity

- **Referential integrity constraint** - Defines a foreign key that references the primary key of another table.
  - If a foreign key contains a value that is not `NULL`, that value must be present in some tuple in the relation containing the referenced primary key.
- Example: `Workson` contains two foreign keys:
  - `workson.essn` references `employee.ssn`
  - `workson.pno` references `project.pnumber`
- Specify foreign keys using `FOREIGN KEY` syntax:

```
FOREIGN KEY (essn) REFERENCES employee(ssn)
```

SQL Referential Integrity

- The `CREATE TABLE` command for the `workson` relation:

```
CREATE TABLE workson (
    essn   CHAR(9),
   pno  INT(4),
   hoursDECIMAL(4,1),
   PRIMARY KEY (essn,pno),
   FOREIGN KEY (essn) REFERENCES
employee(ssn),
   FOREIGN KEY (pno) REFERENCES
project(pnumber)
);
```

SQL Referential Integrity and Updates

- When you try to `INSERT` or `UPDATE` a **row in a relation containing a foreign key** (e.g. `workson`) that operation is rejected if it violates referential integrity.

- When you `UPDATE` or `DELETE` a **row in the primary key relation** (e.g. `emp` or `proj`), you have the option on what happens to the values in the foreign key relation (`workson`):
  - 1) `CASCADE` - Delete (update) values in foreign key relation when primary key relation has rows deleted (updated).
  - 2) `SET NULL` - Set foreign key fields to `NULL` when corresponding primary key relation row is deleted.
  - 3) `SET DEFAULT` - Set foreign key values to their default value (if defined).
  - 4) `NO ACTION` - Reject the request on the parent table.

SQL Referential Integrity Example (2)

```
CREATE TABLE workson (
   essn  CHAR(9),
   pno   INT(4),
   hours DECIMAL (4,1),
   PRIMARY KEY (essn,pno),
   FOREIGN KEY (essn) REFERENCES employee(ssn)
                              ON DELETE NO ACTION
                              ON UPDATE CASCADE,
   FOREIGN KEY (pno) REFERENCES project(pnumber)
                              ON DELETE NO ACTION
                              ON UPDATE CASCADE
);
```

You don't want to delete an employee who is still
Working on a project…delete from WorksOn first

SQL CREATE TABLE Example

- The CREATE TABLE command for the Emp relation:

```
CREATE TABLE employee (
   ssn      CHAR(9),
   lname    VARCHAR(15) NOT NULL,
   …
   superssn CHAR(9),
   dno      INT(4),
   PRIMARY KEY (eno),
   FOREIGN KEY (dno) REFERENCES department(dnum)
         ON DELETE SET NULL ON UPDATE CASCADE,
   FOREIGN KEY (superssn) REFERENCES employee(ssn)
         ON DELETE SET DEFAULT ON UPDATE CASCADE,
);
```

If a department is deleted, do not fire the employee
IF supervisor is deleted, set to default supervisor

Domain Constraints SQL

- Name should not be NULL
- Age > 10 (restrict values in that domain)
- Other constraints…
  - Can specify SQL query

```
CREATE TABLE Students
        (sid CHAR(20),
         name: CHAR(20) NOT NULL,
         login CHAR(10),
         age INTEGER,
         gpa: REAL,
        CHECK (age > 10) ) ;
```

SQL `CREATE TABLE` Full Syntax

- Full syntax of `CREATE TABLE` statement:

```
CREATE TABLE tableName (
  { attrName attrType  [NOT NULL] [UNIQUE] [PRIMARY KEY]
      [DEFAULT value] [CHECK (condition)] }
   [PRIMARY KEY (colList)]
   {[FOREIGN KEY (colList) REFERENCES tbl [(colList)],
      [ON UPDATE action]
      [ON DELETE action] ] }
   {[CHECK (condition)] }
);
```

Important: MySQL does not support CHECK operator
Implement this using TRIGGERS
- Will return to this in a few weeks

Database Updates

- Database updates such as inserting rows, deleting rows, and updating rows are performed using their own statements.
- INSERT
- UPDATE
- DELETE

Database Updates

- Insert is performed using the `INSERT` command:

```
INSERT INTO tableName [(column list)]
VALUES (data value list)
```

- Examples:

```
INSERT INTO employee VALUES
('James','E','Borg','888665555','1927-11-10',
        '450 Stone, Houston, TX','M',55000,null,null);

INSERT INTO project (pno, pname)
   VALUES ('P6','Programming');
```

Note: If column list is omitted, values must be specified in order they were created in the table. If any columns are omitted from the list, they are set to `NULL`.

Changing/Deleting Tables/Schema…Read on your own

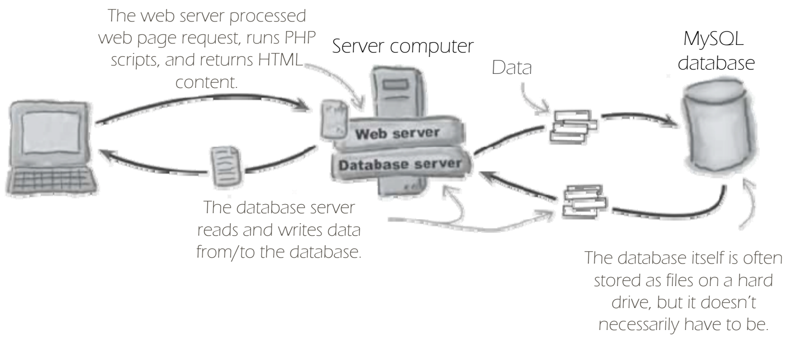
- The **`ALTER TABLE`** command can be used to change an existing table. This is useful when the table already contains data and you want to add or remove a column or constraint.
  - DB vendors may support only parts of `ALTER TABLE` or may allow additional changes including changing the data type of a column.
- The command **`DROP TABLE`** is used to delete the table definition and all data from the database:

```
DROP TABLE tableName [RESTRICT | CASCADE];
```

DDL Summary

- **SQL contains a data definition language that allows you to `CREATE`, `ALTER`, and `DROP` database objects such as tables, triggers, indexes, schemas, and views.**
- **Constraints are used to preserve the integrity of the database:**
  - **`CHECK` can be used to validate attribute values.**
  - **Entity Integrity constraint - The primary key of a table must contain a unique, non-null value for each row.**
  - **Referential integrity constraint - Defines a foreign key that references a unique key of another table.**
- **`INSERT`, `DELETE`, and `UPDATE` commands modify the data stored within the database.**

Next: Module 2 – Getting started with MySQL…



Connecting to mySQL on gwupyterhub

- Use your GW netID to connect to the gwupyterhub.seas.gwu.edu server

```
ssh -Y GWnetID@gwupyterhub.seas.gwu.edu
```

- Login into MySQL

```
mysql -u GWnetID -p
```

NOTE: use your GW NetID, WITH the password CSCI2541_sp20

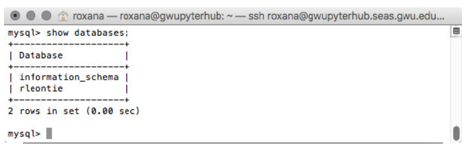
- Reset your password

```
SET PASSWORD FOR 'GWNetID'@'localhost'='NEWPASSWORD';
```

MySQL Database

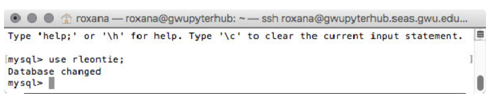
- An existing database is available for your use

```
show databases;
```



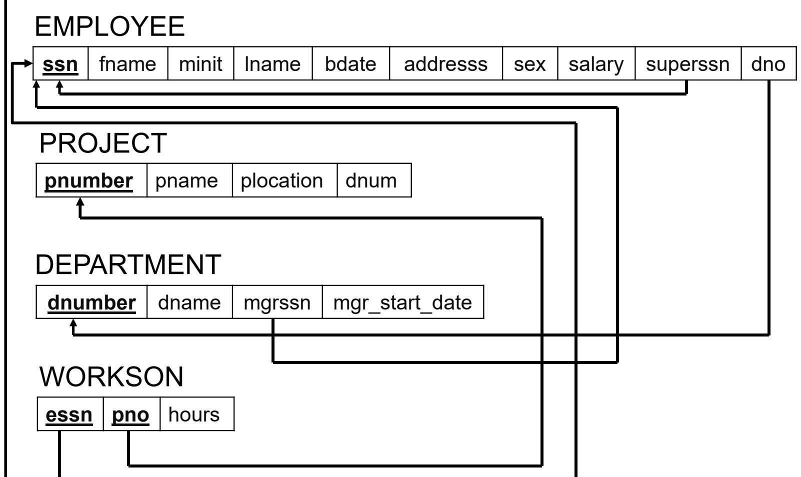
- To use your database:

```
use database_name;
```



NOTE: use your GW NetID for database name

Employee Relational Database Schema (simple)



MySQL table creation

- Syntax to create a table example

```
CREATE TABLE works_on (
  essn   char(9),
  pno    int(4),
  hours  decimal(4,1),
  primary key (essn,pno),
  foreign key (essn) references employee(ssn),
  foreign key (pno) references project(pnumber)
);
```

Note!
`primary key` and `foreign key` denote the constraints

Note!
Tables `employee` and `project` should to be created before!

MySQL basic Data Types

char(n)	Fixed length character string of length n (max 255)
varchar(n)	Variable length character string (max 255)
date	holds a date field (28-Jan-2013)
decimal(n,d)	real numbers occupying up to n spaces with d digits after the decimal point
int(n)	integer with up to n digits

MySQL Table Creation - Example

```
mysql> CREATE TABLE project (
    ->     pname      varchar(25) not null,
    ->     pnumber    int(4),
    ->     plocation  varchar(15),
    ->     dnum       int(4) not null,
    ->     primary key (pnumber),
    ->     unique (pname),
    ->     foreign key (dnum) references department(dnumber)
    -> );
```

```
show tables;
```

Shows all the tables in the database you are currently using.

MySQL table operations:

- Show structure

```
describe tableName;
```

```
mysql> describe employee;
+----------+---------------+------+-----+---------+-------+
| Field    | Type          | Null | Key | Default | Extra |
+----------+---------------+------+-----+---------+-------+
| fname    | varchar(15)   | NO   |     | NULL    |       |
| minit    | varchar(1)    | YES  |     | NULL    |       |
| lname    | varchar(15)   | NO   |     | NULL    |       |
| ssn      | char(9)       | NO   | PRI |         |       |
| bdate    | date          | YES  |     | NULL    |       |
| address  | varchar(50)   | YES  |     | NULL    |       |
| sex      | char(1)       | YES  |     | NULL    |       |
| salary   | decimal(10,2) | YES  |     | NULL    |       |
| superssn | char(9)       | YES  | MUL | NULL    |       |
| dno      | int(4)        | YES  | MUL | NULL    |       |
+----------+---------------+------+-----+---------+-------+
10 rows in set (0.01 sec)

mysql>
```

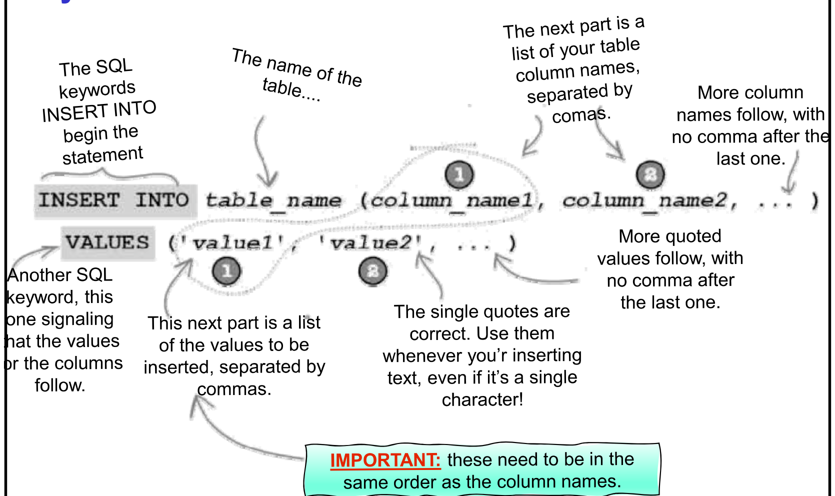
- Modify structure

```
ALTER TABLE employee ADD(bdate date);
ALTER TABLE project DROP column plocation;
ALTER TABLE department MODIFY COLUMN dname varchar(2);
ALTER TABLE employee
ADD foreign key (dno) references department(dnumber);
```

- Remove table

```
DROP TABLE locations;
```

MySQL: INSERT



MySQL: INSERT - Example

```
mysql> describe employee;
+----------+---------------+------+-----+---------+-------+
| Field    | Type          | Null | Key | Default | Extra |
+----------+---------------+------+-----+---------+-------+
| fname    | varchar(15)   | NO   |     | NULL    |       |
| minit    | varchar(1)    | YES  |     | NULL    |       |
| lname    | varchar(15)   | NO   |     | NULL    |       |
| ssn      | char(9)       | NO   | PRI |         |       |
| bdate    | date          | YES  |     | NULL    |       |
| address  | varchar(50)   | YES  |     | NULL    |       |
| sex      | char(1)       | YES  |     | NULL    |       |
| salary   | decimal(10,2) | YES  |     | NULL    |       |
| superssn | char(9)       | YES  | MUL | NULL    |       |
| dno      | int(4)        | YES  | MUL | NULL    |       |
+----------+---------------+------+-----+---------+-------+
10 rows in set (0.00 sec)

mysql> INSERT INTO employee VALUES
    ->    ('James','E','Borg','888665555','1927-11-10','450 Stone, Houston, TX',
'M',55000,null,null);
Query OK, 1 row affected (0.01 sec)

mysql>
```

Check that data was added: SELECT

Follow SELECT with a list of the columns you want data for.

A SELECT always take place with respect to a specific table, not a database in general.

`SELECT columns FROM table_name`

The FROM part of a SELECT statement is how SELECT knows what table we'll be selecting data from.

```
roxanaleontie — ssh roxana@seasdb.seas.gwu.edu — 116×10

mysql> SELECT * from employee;
+-------+-------+-------+-----------+------------+------------------------+-----+----------+----------+------+
| fname | minit | lname | ssn       | bdate      | address                | sex | salary   | superssn | dno  |
+-------+-------+-------+-----------+------------+------------------------+-----+----------+----------+------+
| James | E     | Borg  | 888665555 | 1927-11-10 | 450 Stone, Houston, TX | M   | 55000.00 | NULL     | NULL |
+-------+-------+-------+-----------+------------+------------------------+-----+----------+----------+------+
1 row in set (0.00 sec)

mysql> 
```

In class exercise:

https://classroom.github.com/a/wBGgagip

- Create all tables for the employee schema described im Module 2
  - For each table creation record the create statements and show the description
- Run the script provided to populate the tables.
- Show all the entries in each table.

NOTE:

Because of the connectivity of these tables, when creating them, run the following command:

`SET FOREIGN_KEY_CHECKS = 0;`

After table creation:

`SET FOREIGN_KEY_CHECKS = 1;`

You will also need to alter the tables and add the foreign keys relationship after the tables dependent are created.

Next: Module 3- Querying in SQL

- Querying the database
  - SQL Data Manipulation language
- Today….Simple commands
  - Equivalent to Relational Algebra

Basic SQL Query

SELECT	[DISTINCT] *attribute-list*
FROM	*relation-list*
WHERE	*qualification/predicate :*

- *relation-list* A list of relation names (possibly with a *range-variable, i.e., tuple variable,* after each name).
- *attribute-list* A list of attributes of relations in *relation-list*
- *Qualification/predicate* Comparisons (Attr *op* const or Attr1 *op* Attr2, where *op* is one of $<, >, =, \leq, \geq, \neq$) combined using AND, OR and NOT.
- DISTINCT is an optional keyword indicating that the answer should not contain duplicates. Default is that duplicates are *not* eliminated!
- To select all attributes in result, we use *

SQL and Relational Algebra

- The `SELECT` statement can be mapped directly to relational algebra.

SELECT $A_1, A_2, \ldots, A_n$ *this is projection π*
FROM $R_1, R_2, \ldots, R_m$ *this is Cartesian product ×*
WHERE P *this is the selection op σ*

- is equivalent to:

$$\Pi_{A_1, A_2, \ldots, A_n}(\sigma_P(R_1 \times R_2 \times \ldots \times R_m))$$

- If we don't want to project, then SELECT *

Conceptual Evaluation Strategy

- *Semantics* of an SQL query defined in terms of the following conceptual evaluation strategy:
- Compute the cross-product of *relation-list*.
- Discard resulting tuples if they fail *predicate qualifications*.
- Delete attributes that are not in *target attribute-list*.
  - If DISTINCT is specified, eliminate duplicate rows.
    - SQL allows duplicates in relations (unlike Rel. Algebra)
- This strategy is probably the least efficient way to compute a query! An optimizer will find more efficient strategies to compute *the same answers*.

Simple SQL Query

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

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

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks

"selection"

Algorithm: Scan each tuple in table and check if matches condition in `WHERE` clause.

Simple SQL Query

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

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

PName	Price	Manufacturer
SingleTouch	$149.99	Canon
MultiTouch	$203.99	Hitachi

"selection" and "projection"

Duplicates in SQL

SQL returns 'bag of words'

duplicates allowed in contrast to relational algebra

To remove duplicates use DISTINCT clause:

```
SELECT DISTINCT title
FROM emp;
```

Eliminating Duplicates

```
SELECT   DISTINCT category
   FROM     Product
```

Category
Gadgets
Photography
Household

Compare to:

```
SELECT   category
 FROM    Product
```

Category
Gadgets
Gadgets
Photography
Household

Expressions and Strings

```
SELECT S.sid
FROM Students S
WHERE S.name LIKE '%Sam%'
```

- Illustrates use of arithmetic expressions and string pattern matching: *Find students whose name includes 'Sam'.*
- LIKE is used for string matching. `_' stands for any one character and `%' stands for 0 or more arbitrary characters.

- Find students whose name begins with S and at least three characters:
  - Replace with 'S_ _ %'

Ordering Result Data

- The query result returned is not ordered on any attribute by default. We can order the data using the **ORDER BY** clause:
- STUDENTS [sid, name]

```
SELECT   name
FROM     students
ORDER BY name ASC
```

- 'ASC' sorts the data in ascending order, and 'DESC' sorts it in descending order. The default is 'ASC'.
- The order of sorted attributes specified by the 'sort key' (name in above example) NULL is normally treated as less than all non-null values.

Ordering Result Data using multiple attributes

- Can define sort on major and minor sort keys
  - Corresponds to “filing order” of k-tuples

emp: [ssn, ename, salary]

```
SELECT    ename, salary
FROM      emp
WHERE     salary > 30000
ORDER BY salary DESC, ename ASC
```

ename	salary
Adam	100000
Jill	100000
Bill	80000

- The order of sorted attributes is significant. The first attribute specified is sorted on first, then the second attribute is used to break any ties, etc.

Bank Database Schema

Branch

Branch_Name	Assets	Branch_City

Loan

CustID	**LoanNo**	Amount	Branch_Name

Deposit

CustID	**AccNo**	Balance	Branch_Name

Customer

CustID	Name	Street	City	Zip

Schema of Bank DB

- Customer (CustID, Name, street,city,zip)
  - Key is CustID of type int (?)
  - Name, street, city can vary in length – varchar(20
  - CustID and zip can be integer
- Deposit (CustID, Acct-num, balance,Branch-name)
  - Acct-Num is key, type int (?)
  - CustID foreign key references Customer
  - Branch-name is varchar(20) references Branch
  - Balance is a real number..i.e., decimal
  - Customer can have many accounts; cannot have joint accounts
- Loan (CustID, Loan-num, Amount, Branch-name)
- Branch (Branch-name, assets, Branch-city)

IN CLASS EXERCISE

- Follow the instructions in your GitHub repository to populate the Bank database.
- TODO: move 4 queries here

Joins in SQL

- Multiple tables can be queried in a single SQL statement by listing them in the `FROM` clause.
  - Note that if you do not specify any join condition to relate them in the `WHERE` clause, you get a *cross product* of the tables.

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
```

Join between Product and Company

Joins

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

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

Company

Cname	StockPrice	Country
GizmoWorks	25	USA
Canon	65	Japan
Hitachi	15	Japan

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

PName	Price
SingleTouch	$149.99

Renaming and Aliasing

- Does the job of rename operator ρ in relational algebra
- Often it is useful to be able to rename an attribute in the final result (especially when using calculated fields). Renaming is accomplished using the keyword **AS**:

```
SELECT lname, salary AS pay
FROM   employee
WHERE  dno=5;
```

Result

lname	pay
Lee	100000.00
Smith	60000.50
Lee	90000.00

Note: AS keyword is optional.

Renaming…Using Tuple/Range variables

- Concept of tuple/range variables borrowed from relational calculus
  - Tuple *t* of type *R*: $t \in R$
  - *What about* $x \in R$, $y \in R$
- It performs the job of the rename operator from relational algebra
  - **One variable with name *x* and one with name *y, BOTH of type R***
- Need to worry about scope of tuple variables when we have nested queries

Aliasing to remove ambiguity…The easy case:

Person(pname, address, worksfor)
Company(cname, address)

```
SELECT   DISTINCT pname, address
FROM     Person, Company
WHERE    worksfor = cname
```

Which address ?

```
SELECT   DISTINCT Person.pname, Company.address
FROM     Person, Company  /*named field notation
WHERE    Person.worksfor = Company.cname
```

```
SELECT   DISTINCT x.pname, y.address
FROM     Person AS x, Company AS y /* aliasing
WHERE    x.worksfor = y.cname
```

Tuple Variables

Person(pname, address, worksfor)
Company(cname, address)

```
SELECT   DISTINCT P.pname,C.address
FROM     Person P, Company C
 WHERE   P.worksfor = C.cname;
```

x is a copy of Person, y is a copy of Company
P is a variable of ‘type’ Person C is a variable of ‘type’ Company

Renaming: Joining table with itself

- Aliases/Tuple variables must be used when relation has to be ‘joined’ with itself – i.e., two or more copies of the same table are needed. Using ***aliases*** allows you to uniquely identify what table you are talking about.

Example: Return last names of employees and their managers.

```
SELECT E.lname, M.lname
FROM   employee E, employee M
WHERE E.superssn = M.ssn;
```

- E is a variable of type Employee, and denotes an employee
- M is a variable of type Employee, and denotes (will bind to) values of supervisor

Meaning (Semantics) of SQL Queries with tuple variables

SELECT $a_1, a_2, \ldots, a_k$
FROM $R_1\ x_1, R_2\ x_2, \ldots, R_n\ x_n$
WHERE Conditions

Answer = {}
for x_1 **in** R_1 **do**
    for x_2 **in** R_2 **do**
        …..
            for x_n **in** R_n **do**
                if Conditions
                    then Answer = Answer $\cup$ $\{(a_1,\ldots,a_k)\}$
return Answer

Tuple variables

- Find students who are taking the same course as Sam with sid=1234.
- Need to access Takes table twice
  - Once to extract courses (X) taken by Sam with ID=1234
  - Second time to find students who are taking these X courses
- Define two "variables" A,B of 'type' Takes
  - B is variable that corresponds ID 1234 and its cid field is equal to "*X*"
  - A is a variable whose CID is equal to "*X*"
- SELECT B.sid
- FROM Takes A, Takes B
- WHERE A.cid = B.cid;

More SQL

- Set opertions
- Aggregate operators
- GroupBy
- ……

- Next week

Next: Module 4- Test your querying skills!

- Step 1: 15 minutes
  - Do NOT code…
  - Work at your table to discuss solutions/queries – do not write down code

- Step 2: Code your queries and demo to the instructors
  - And submit solutions/code

Questions:

1. Find all rows in deposit where balance is greater than $1000
2. Find names of customers whose name is the same as the street they live on.
3. Find names and IDs of customers, ordered by name, whose name begins with a C
4. Find IDs of all customers who have Loans between 1200 and 2500
5. Find names of all customers who have Loans between 1200 and 2500.
6. Find the names and IDs of customers who live on the same street as customer(s) named Lennon
7. Find names of customers who have an account at the same branch as a customer named Lennon.