INFSCI 1022
Database Management Systems

Today’s Evil Plan

- Writing queries
  - Create
  - Drop
  - Insert
  - Update
  - Delete
  - Select

Writing Queries

- ✓ SELECT
- ✓ INSERT
- ✓ UPDATE
- ✓ DELETE

Writing Queries

- ✓ CREATE
- ✓ DROP
- ✓ INSERT
- ✓ UPDATE
- ✓ DELETE
- ✓ SELECT

SQL

- SQL - Structured Query Language, a special-purpose programming language designed for managing data held in a relational database
- SQL is almost English; it's made up largely of English words, put together into strings of words that sound similar to English sentences.

Query Types

- The first word of each query is its name, which is an action word (a verb) that tells DMBS what you want to do.
  - CREATE – creates a new table or a schema
  - DROP – drops an existing table or a schema
  - SELECT – retrieves data from a table or a set of tables
  - INSERT – creates a new record in a single table
  - UPDATE – updates in a single table
  - DELETE – deletes/removes a record from a single table

Writing Queries

- ✓ **CREATE**
- ✓ DROP
- ✓ INSERT
- ✓ UPDATE
- ✓ DELETE
- ✓ SELECT

CREATE DATABASE

CREATE DATABASE [database name];

CREATE DATABASE

```
CREATE DATABASE movie_tracker;
```

USE DATABASE

USE [database name] statement will tell MySQL that all of your queries should be executed against the specified database.

USE DATABASE

```
USE movie_tracker;
```

SHOW TABLES

SHOW TABLES statement will give you a list of all tables in your database;

CREATE TABLE

CREATE TABLE statement is used to specify the logical layout of a table and to create a database table.

CREATE TABLE

```
CREATE TABLE [TEMPORARY] TABLE [IF NOT EXISTS] tbl_name
    (create_definition,...)
    [table_options]
    [partition_options]
```

http://dev.mysql.com/doc/refman/5.1/en/create-table.html

CREATE TABLE

```
CREATE TABLE  movie (
    movie_id INT,
    title VARCHAR(200),
    budget DOUBLE,
    release_date DATETIME
);
```

CREATE TABLE

```
CREATE TABLE  movie (
    movie_id INT NOT NULL,
    title VARCHAR(200) NOT NULL,
    budget DOUBLE NOT NULL,
    release_date DATETIME NOT NULL
);
```

CREATE TABLE

```
CREATE TABLE  movie (
    movie_id INT PRIMARY KEY NOT NULL,
    title VARCHAR(200) NOT NULL,
    budget DOUBLE NOT NULL,
    release_date DATETIME NOT NULL
);
```

CREATE TABLE

```
CREATE TABLE  movie (
    movie_id INT PRIMARY KEY NOT NULL AUTO_INCREMENT,
    title VARCHAR(200) NOT NULL,
    budget DOUBLE NOT NULL,
    release_date DATETIME NOT NULL
);
```

NUMERIC DATA TYPES

- Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT
- Fixed-Point Types (Exact Value) - DECIMAL, NUMERIC
- Floating-Point Types (Approximate Value) - FLOAT, DOUBLE
- Bit-Value Type - BIT

http://dev.mysql.com/doc/refman/5.6/en/numeric-types.html

STRING DATA TYPES

- CHAR and VARCHAR Types
- BINARY and VARBINARY Types
- BLOB and TEXT Types

http://dev.mysql.com/doc/refman/5.6/en/string-types.html

DATE/TIME DATA TYPES

- DATE, DATETIME, and TIMESTAMP Types
- TIME Type
- YEAR Type

http://dev.mysql.com/doc/refman/5.6/en/date-and-time-types.html

Writing Queries

- ✓ CREATE
- ✓ **DROP**
- ✓ INSERT
- ✓ UPDATE
- ✓ DELETE
- ✓ SELECT

DROP STATEMENT

- **DROP TABLE** [table name];

DROP STATEMENT

- **DROP TABLE** movies;

CREATING FOREIGN KEYS

```
CREATE TABLE characters (
    character_id INT PRIMARY KEY NOT NULL AUTO_INCREMENT,
    character_name VARCHAR(150) NOT NULL,
    character_description VARCHAR(4000),
    movie_id INT,
    FOREIGN KEY (movie_id)
    REFERENCES movies(movie_id)
    ON DELETE CASCADE
    ON UPDATE CASCADE
);
```

CREATING FOREIGN KEYS

```
ALTER TABLE characters
ADD CONSTRAINT
FOREIGN KEY (movie_id)
REFERENCES movies(movie_id)
ON DELETE CASCADE
ON UPDATE CASCADE;
```

Writing Queries

- ✓ CREATE
- ✓ DROP
- ✓ **INSERT**
- ✓ UPDATE
- ✓ DELETE
- ✓ SELECT

INSERT

```
INSERT INTO table_name ( field1, field2,...fieldN )
                VALUES
                  ( value1, value2,...valueN );
```

INSERT

```
INSERT INTO classicmodels.payments
(customerNumber, checkNumber, paymentDate, amount)
VALUES
(103, 1, ‘2014-10-10’, 4000);
```

Writing Queries

- ✓ CREATE
- ✓ DROP
- ✓ INSERT
- ✓ **UPDATE**
- ✓ DELETE
- ✓ SELECT

UPDATE

UPDATE table_name **SET** field1=new-value1, field2=new-value2 [WHERE Clause]

UPDATE

```
UPDATE classicmodels.payments
SET amount = 10000
WHERE customerNumber = 103
AND checkNumber = 1;
```

UPDATE

```
UPDATE classicmodels.payments
SET amount = 10000, checkNumber = ‘XXXXXX’
WHERE customerNumber = 103
AND checkNumber = 1;
```

DELETE

DELETE FROM table_name [WHERE Clause]

Writing Queries

- ✓ CREATE
- ✓ DROP
- ✓ INSERT
- ✓ UPDATE
- ✓ **DELETE**
- ✓ SELECT

DELETE

```
DELETE FROM classicmodels.payments
WHERE customerNumber = 103
AND checkNumber = 1;
```

TRUNCATE

TRUNCATE TABLE [table name];

- **DELETE** statement deletes individual row(s) in a database. You can specify which rows to delete using the WHERE clause
- **TRUNCATE** statement deletes ALL rows – it completely empties a table.

TRUNCATE

TRUNCATE TABLE movies;

Writing Queries

- ✓ CREATE
- ✓ DROP
- ✓ INSERT
- ✓ UPDATE
- ✓ DELETE
- ✓ **SELECT**

SELECT Queries

Database (schema) name

Name of the table from which you are retrieving data

```
SELECT * FROM classicmodels.offices;
```

SELECT keyword

* means selecting ALL columns from a table

FROM keyword – specifies the start of the FROM clause

SELECT Queries

Selecting a list of columns

```
SELECT officeCode, city
FROM classicmodels.offices;
```

It’s a good practice to end a query with a semicolon

Query Clauses

Clauses - constituent components of statements and queries.

- **FROM**
- **WHERE**
- GROUP BY
- HAVING
- **ORDER BY**
- **LIMIT**

FROM

- Indicates the table(s) from which data is to be retrieved.

SELECT * **FROM classicmodels.offices**

WHERE

- Includes a comparison predicate, which restricts the rows returned by the query.
- The WHERE clause eliminates all rows from the result set for which the comparison predicate does not evaluate to True.

```
SELECT * FROM classicmodels.offices
WHERE city = 'Boston';
```

AND OPERATOR

- ***AND*** operator is used in WHERE clauses
- Allows to limit query results be comparing values against multiple fields

```
SELECT * FROM classicmodels.offices
WHERE city = 'Boston' AND territory = 'NA';
```

ORDER BY

- Identifies which columns are used to sort the resulting data, and in which direction they should be sorted (options are ascending or descending).
- Without an ORDER BY clause, the order of rows returned by an SQL query is undefined.

```
SELECT * FROM classicmodels.offices
ORDER BY city DESC
```

ORDER OF CLAUSES

- CLAUSES must appear in the following order
  - **FROM**
  - **WHERE**
  - **GROUP BY**
  - **HAVING**
  - **ORDER BY**
- Not all clauses must appear in a query – **FROM** clause is the only one that's required

Operators

Operator	Description	Example
`=`	Equal to	`Author = 'Alcott'`
`<>`	Not equal to (most DBMS also accept `!=` instead of `<>`)	`Dept <> 'Sales'`
`>`	Greater than	`Hire_Date > '2012-01-31'`
`<`	Less than	`Bonus < 50000.00`
`>=`	Greater than or equal	`Dependents >= 2`
`<=`	Less than or equal	`Rate <= 0.05`
`BETWEEN`	Between an inclusive range	`Cost BETWEEN 100.00 AND 500.00`
`LIKE`	Match a character pattern	`First_Name LIKE 'Will%'`
`IN`	Equal to one of multiple possible values	`DeptCode IN (101, 103, 209)`
`IS` *or* `IS NOT`	Compare to null (missing data)	`Address IS NOT NULL`

Homework

- Watch the first four sections of “Foundations of Programming: Databases” on Lynda.com
  - Go to lynda.pitt.edu and login using your Pitt credentials
  - Copy the following URL and paste it into your browser’s address bar: http://www.lynda.com/Programming-tutorials/Foundations-Programming-Databases/112585-2.html
- Recommended SQLZoo Tutorials: http://sqlzoo.net/
- Complete homework assignment posted on Canvas