SQL

CS 1270
Agenda

1. SQL
   a. DDL
   b. DML
      i. Basic SQL
      ii. Sub queries
      iii. Advanced SQL
SQL?

SQL is a language for writing DDL (Data definition language) and DML (data manipulation language)

- Design how data is stored at a logical/functional level
- Communicate desired data query/manipulation
- DBMS manages underlying logic for how to best store/compute/access the data
  - More on this later!!
DDL

Define table structure including:

- Data types
- Keys (primary, foreign)
- Data Constraints (NOT NULL, UNIQUE, > 0…)
- Sortkeys (Not much in this class)
- Distribution keys (for Distributed DBs) (Not much in this class)
CREATE TABLE

CREATE TABLE <name_of_relation> (  
   <attribute_name>  <data_type>  <constraint>,
   ...
   <integrity_constraint>,
   ...
);

Data Types

\textbf{char(n)}: A fixed-length character string with user-specified length \( n \).

\textbf{varchar(n)}: A variable-length character string with user-specified maximum length \( n \).

\textbf{int}: An integer (a finite subset of the integers that is machine dependent).

\textbf{numeric(p, d)}: A fixed-point number with user-specified precision. The number consists of \( p \) digits (plus a sign), and \( d \) of the \( p \) digits are to the right of the decimal point. Thus, numeric(3,1) allows 44.5 to be stored exactly, but neither 444.5 or 0.32 can be stored exactly in a field of this type.

\textbf{real, double precision}: Floating-point and double-precision floating-point numbers with machine-dependent precision.

\textbf{float(n)}: A floating-point number, with precision of at least \( n \) digits.
Integrity Constraints

- **Primary key**: `PRIMARY KEY(<attribute_name>, <attribute_name>, ...)`
  - Primary key of a table; must be non-null and unique
- **Foreign Key**: `FOREIGN KEY(<attribute_name>, <attribute_name>, ...) REFERENCES <table_name>`
  - References the primary key of another relation
- **non-null**: `<attr> NOT NULL`
- **Unique**: `<attr> UNIQUE`
  - All values must be unique (*can* include nulls)
  - Also can do: `UNIQUE(<attr1>, <attr2> ...)`
Example

CREATE TABLE department (  
department_name varchar (20),  
building varchar (15) NOT NULL,  
class_id INTEGER,  
budget numeric (12,2) CHECK (budget > 0),  
PRIMARY KEY (department_name, class_id));
Practice!

Try writing DDL for the following 3 tables:

`student(student_id, graduation_year, gpa)`
`class(class_id, course_name)`
`takes(student_id, class_id, year)`
Answer

CREATE TABLE student ( 
student_id INTEGER, 
graduation_year DATE, 
gpa DOUBLE CHECK (gpa >= 0.0 AND gpa <= 4.0) 
PRIMARY KEY (student_id));

CREATE TABLE course( 
course_id INTEGER, 
course_name VARCHAR(256) 
PRIMARY KEY course_id); 

CREATE TABLE takes( 
student_id INTEGER, 
course_id INTEGER, 
year DATE, 
PRIMARY KEY (student_id, course_id, year) 
FOREIGN KEY (student_id) REFERENCES student, 
FOREIGN KEY (course_id) REFERENCES course);
Other DDL

- **ALTER TABLE**
  - Allows you to change table structure. What you can change depends on DBMS. May allow you to drop columns, or add columns, change table name, or more.

- **DROP TABLE**
  - Drops a table deleting all data and its schema

- **TRUNCATE TABLE**
  - Clear all data from a table, but **preserve** the table schema/definition (now empty)

- **INSERT INTO**
  - Load data into a table
  - May take on a list of tuples: `((col1,col2,col3),(col1,col2,col3))`
  - May also take on another select: `SELECT col1, col2, col3 FROM ...`
  - Behaves differently for each different DBMS
SQL DML Core

SELECT

WHERE

JOIN

Aggregates / GROUP BY / ORDER BY
SELECT

- Way to retrieve data from a table
- SQL does contain duplicates
  - Unless you use the word DISTINCT which removes duplicates
- Returns a relation
- * implies all columns in relations
- Can apply arithmetic and more to columns as read in
- Commands formatted like:

```
SELECT <DISTINCT>
<attribute_name>*
FROM <relation>*
```
SELECT Examples

student(student_id, graduation_year, gpa)
class(class_id, course_name)
takes(student_id, class_id, year)

SELECT Student_id
FROM student

SELECT * 
FROM student

SELECT Student_id, graduation_year
FROM student

SELECT Gpa + 1.0 AS better_gpa
FROM student

SELECT CAST(class_id  AS VARCHAR(256)) || course_name AS id_name
FROM class
WHERE

- Addition to a select statement
- Takes form: `WHERE <predicate>`
- Predicate may or may not check column values
  - `<attr_1> = <attr_2>`
  - `<attr_1> <= <attr_2>` AND
WHERE Examples

student(student_id, graduation_year, gpa)
class(class_id, course_name)
takes(student_id, class_id, year)

SELECT student_id
FROM student
WHERE graduation_year = '2020'

SELECT *
FROM student
WHERE CAST (student_id AS FLOAT) = gpa

SELECT student_id
FROM class
WHERE class_id = 127 AND course_name = 'best_course'
Basic Logical Operators in SQL

<attr> AND <attr>

<attr> OR <attr>

<attr> IS NULL

<attr> IS NOT NULL

Many more, and some are DBMS dependent
SQL JOINS

- Basic way to combine tables on shared columns
- Different Joins
  - INNER JOIN (Natural Join)
    - Return rows for which predicate is true in left and right tables
  - LEFT JOIN
    - Preserve all rows in LEFT table and join any row in right table for which predicate is true
  - RIGHT JOIN
    - Preserve all rows in RIGHT table and join any row in left table for which predicate is true
  - OUTER JOIN (Not in SQLite)
    - Preserve all rows in both RIGHT and LEFT tables and join any rows for which predicate is true
JOIN Examples and Syntax

SELECT
...
FROM <table_1>
(LEFT | RIGHT | INNER) JOIN <table_2>
ON <table_1>.col_1 = <table_2>.col_1
JOIN Examples

student(student_id, graduation_year, gpa)
class(class_id, course_name)
takes(student_id, class_id, year)

SELECT
s.student_id,
t.class_id,
t.year
FROM student s
INNER JOIN takes t
ON s.student_id = t.student_id

SELECT
c.course_name,
t.class_id,
t.year
FROM class c
INNER JOIN takes t
ON c.class_id = t.class_id
AND t.year = '2018'
JOIN question

SELECT ...
FROM <table_1>
(LEFT | RIGHT | INNER) JOIN <table_2>
ON ?

What happens if we joined on:

1. <table_1>.col = <constant>
2. 1 = 1
3. 0 = 1
Aggregates

Allow you to calculate aggregates for columns (or a subset of columns)

- **AVG**: average value
- **MIN**: minimum value
- **MAX**: maximum value
- **SUM**: sum of values
- **COUNT**: number of values

**SELECT**

AGG (<DISTINCT> (<attribute_name>, ...))

**FROM** <relation>
GROUP/ORDER BY

- How to aggregate maintaining columns over which the aggregate is calculated
- Required if aggregating over a subset of columns
- Same syntax for order by, but sorts the columns

```
SELECT
Col1, ...
AGG (<DISTINCT> (<attribute_name>, ...))
FROM <relation>
GROUP BY col1,...
```

```
SELECT
Col1, ...
FROM <relation>
ORDER BY col1,... <DESC/ASC>
```
Aggregate Examples

\[
\text{student}(\text{student}\_id, \text{graduation}\_year, \text{gpa}) \\
\text{class}(\text{class}\_id, \text{course}\_name) \\
\text{takes}(\text{student}\_id, \text{class}\_id, \text{year})
\]

\[
\begin{align*}
\text{SELECT} & \quad \text{year,} \\
& \quad \text{COUNT(DISTINCT class}\_id) \text{ AS course_count} \\
\& \quad \text{FROM takes} \\
& \quad \text{GROUP BY year}
\end{align*}
\]

\[
\begin{align*}
\text{SELECT} & \quad \text{AVG(gpa) AS avg_gpa,} \\
& \quad \text{MIN(gpa) AS min_gpa,} \\
& \quad \text{MAX(gpa) AS max_gpa} \\
\& \quad \text{FROM student}
\end{align*}
\]

\[
\begin{align*}
\text{SELECT} & \quad \text{class}\_id, \\
& \quad \text{COUNT(DISTINCT student}\_id) \text{ AS total_unique} \\
& \quad \text{FROM student} \\
& \quad \text{GROUP BY class}\_id
\end{align*}
\]
Subqueries

A subquery is using the result of another select (a relation) in order to perform more computation

General format:

```
SELECT <attr> ...
FROM
(SELECT <attr> ...
 FROM <subquery_name>) <subquery_name>
WITH sub_1 AS (SELECT ...),
 sub_2 AS (SELECT ...)
SELECT <attr> ...
FROM sub_1
```
How to use subqueries (or just other relations)

Set functions in SQL:

WHERE \textit{col}_1 \textit{<OP>} (subquery)

\textbf{IN}: \textit{val}_1 \text{ exists in Subquery } S

\textbf{NOT IN}: \textit{val}_1 \text{ does not exist in Subquery } S

\textbf{EXISTS}: \textit{at least 1} \textit{val}_1 \text{ exists in Subquery } S

\textbf{ALL}: \textit{val}_1 \text{ is true for all values in Subquery } S

\textbf{SOME}: \textit{val}_1 \text{ is true for more than 1 value in Subquery } S

\begin{verbatim}
student(\textit{student_id}, \textit{graduation_year}, \textit{gpa})
class(\textit{class_id}, \textit{course_name})
takes(\textit{student_id}, \textit{class_id}, \textit{year})

SELECT \textit{student_id}
FROM student
WHERE \textit{student_id} \text{ IN }
    (SELECT \textit{student_id} FROM takes)
\end{verbatim}
Union behaves its set counterpart does and unions two tables

Unlike RA, you can union tables with differently named columns (not types though). The resulting relation has the top most query column names.

Union removes duplicates, unless you specify UNION ALL which includes duplicates

```
SELECT <col_1 ...>
FROM <relation>
UNION <ALL>
SELECT <col_1 ...>
FROM <relation>
```
Other Fun SQL

SQL constructs that are less commonly used but can be super helpful:

1. **CASE WHEN** `<predicate>` **THEN** `<val>` … **ELSE** `<val>` **END**
   a. **CASE** can be used around any column; that means it can be used in joins, aggregates, where clauses etc.

2. **BELOW ARE NOT TESTED IN CS 127**

3. **Window Functions**
   a. **DML** that allows you to take aggregates at higher granularity; Not offered in all DBMS, mostly used for complex analytical purposes

4. **SORTKEYs**
   a. **DDL** construct that specifies how tables are sorted/indexed

5. **DISTKEYs**
   a. **DDL** construct that specified how data is distributed among nodes in distributed DBMS systems
Interactive Fun!

1. Go to http://www.sqlitetutorial.net/sqlite-sample-database/
2. Click on: Download SQLite Sample Database
3. Extract the Zip
4. You should see chinook.db
5. Go to https://sqliteonline.com/
6. Click File -> Open DB
7. Select chinook.db
8. Play with SQL!
SQL Practice Questions

1. What is the total cost of all of the tracks?
2. What is the total number of songs and total cost of songs for the genres Rock, Metal, and Jazz?
3. What is the most purchased track?
4. Find all the artists names with multiple track genres.
5. Find the First and Last name of the customer who spent the most money.
6. Find the First and Last name of the employee who sold the most money.
How to check answers?

Use SQL!

Compare the result table we have built with yours. If they match then you are good, if not, try to figure out why!

There are many solutions to most of these problems so be creative!
SELECT sum(UnitPrice) AS total_cost FROM tracks;

3680.969999999704
SELECT g.Name, SUM(UnitPrice) AS cost, COUNT(TrackId) AS tracks FROM tracks t
INNER JOIN genres g
ON g.GenreId = t.GenreId
WHERE g.Name IN ('Rock', 'Metal', 'Jazz')
GROUP BY g.Name;

<table>
<thead>
<tr>
<th>Name</th>
<th>cost</th>
<th>tracks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Jazz</td>
<td>128.69999999999997</td>
<td>130</td>
</tr>
<tr>
<td>Metal</td>
<td>370.26000000000019</td>
<td>374</td>
</tr>
<tr>
<td>Rock</td>
<td>1284.03000000000102</td>
<td>1297</td>
</tr>
</tbody>
</table>
WITH sums AS (SELECT TrackId, SUM(Quantity) AS quant FROM invoice_items)

SELECT
s1.TrackId
FROM sums s1
WHERE s1.quant = (SELECT MAX(quant) FROM sums)

Track 3177, with quantity 2240
SELECT artist_name FROM
(SELECT a.artist_name, COUNT (DISTINCT t.GenreId) AS genres FROM tracks t
INNER JOIN
(SELECT art.ArtistId, AlbumId, art.Name AS artist_name FROM artists art
INNER JOIN albums al
ON art.ArtistId = al.ArtistId) a
ON a.AlbumId = t.AlbumId
GROUP BY a.artist_name)
WHERE genres > 1;

21 Artists, Including: Amy Winehouse, Heroes, U2 ...
WITH prices AS (  SELECT  i.InvoiceId,  i.CustomerId,  items.TotalSpend  FROM invoices i  LEFT JOIN (  SELECT  InvoiceId,  SUM(UnitPrice * Quantity) AS TotalSpend  FROM invoice_items  GROUP BY InvoiceId) items  ON items.InvoiceId = i.InvoiceId),
customer_ltd  AS (  SELECT customers.CustomerId, FirstName, LastName, SUM(prices.TotalSpend) AS ltd_spend FROM customers  INNER JOIN prices  ON prices.CustomerId = customers.CustomerId  GROUP BY customers.CustomerId, FirstName, LastName)

SELECT FirstName, LastName FROM customer_ltd  WHERE ltd_spend = (SELECT MAX(ltd_spend) FROM customer_ltd)

Best way to check this is to compare the results in customer_ltd to your equivalent table.

So use the line:
SELECT * FROM customer_ltd

To compare to your table

ANS:
Helena Holý
WITH prices AS (  
SELECT  
i.InvoiceId,  
i.CustomerId,  
items.TotalSpend  
FROM invoices i  
LEFT JOIN (  
SELECT  
    InvoiceId,  
    SUM(UnitPrice * Quantity) AS TotalSpend  
FROM invoice_items  
GROUP BY InvoiceId) items  
ON items.InvoiceId = i.InvoiceId),

support_ltd AS (  
SELECT customers.SupportRepId, SUM(prices.TotalSpend) AS ltd_spend FROM customers  
INNER JOIN prices  
ON prices.CustomerId = customers.CustomerId  
GROUP BY customers.SupportRepId)

SELECT e.FirstName, e.LastName FROM support_ltd s  
INNER JOIN employees e  
ON s.SupportRepId = e.EmployeeId  
WHERE s.ltd_spend = (SELECT MAX(ltd_spend) FROM support_ltd)