CS 1555
www.cs.pitt.edu/~nlf4/cs1555/
Using SQL as a DML
• Adds tuples to a table

• `INSERT INTO <tname> [<attribute_list>] VALUES (<tuple values>)`
Assuming that

- We're inserting into the following tables:

```
CREATE TABLE Students
( ID INTEGER,
  Name VARCHAR(20),
  Ssn CHAR(9) NOT NULL,
  Major VARCHAR(10),
  GPA DECIMAL(3,2),
CONSTRAINT Students_PK PRIMARY KEY (ID),
CONSTRAINT Students_AK UNIQUE (Ssn)
);
```

```
CREATE TABLE Enrollment
( Stud_ID INTEGER,
  Course VARCHAR(10),
CONSTRAINT Enrollment_FK FOREIGN KEY (Stud_ID) REFERENCES Students(ID)
);
```
### INSERT Examples

<table>
<thead>
<tr>
<th>Enrollment</th>
<th>Students</th>
</tr>
</thead>
<tbody>
<tr>
<td>Stud_ID</td>
<td>Course</td>
</tr>
<tr>
<td>546346</td>
<td>Math 422</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
<tr>
<td>ID</td>
<td>Name</td>
</tr>
<tr>
<td>334322</td>
<td>Alice</td>
</tr>
<tr>
<td>546346</td>
<td>Bob</td>
</tr>
</tbody>
</table>

- **INSERT INTO** Students **VALUES** (334322, 'Alice', '1111111', 'CS', 3.45);
- **INSERT INTO** Enrollment **VALUES** (546346, 'Math 422');
- **INSERT INTO** Students **VALUES** (546346, 'Bob');
- **INSERT INTO** Students (ID, Name) **VALUES** (546346, 'Bob');
- **INSERT INTO** Students (ID, Name, Ssn) **VALUES** (546346, 'Bob', '2222222');
- **INSERT INTO** Enrollment **VALUES** (546346, 'Math 422');
Queries: the SQL SELECT statement

```
SELECT [DISTINCT|ALL] <attribute_list>
FROM <table_list>
WHERE <condition>
GROUP BY <attribute_list>
HAVING <gcondition>
ORDER BY <aname> ASC|DESC[, <aname> ASC|DESC ...]
```
Basic SQL queries

- **Condition operations:**
  - `=, <>, <, <=, >, >=`

- **For Students** (ID, Name, Ssn, Major, GPA), write the following relational algebra operations as SQL queries:
  - `σ_{ID 
eq 334322}(Students)`
  - `π_{ID, Name, GPA}(Students)`
ALL is the default

What about DISTINCT?

What are the results of:

- **SELECT** Major
  **FROM** Students;

- $\pi_{\text{Major}}(\text{Students})$

- **SELECT DISTINCT** Major
  **FROM** Students;

<table>
<thead>
<tr>
<th>Name</th>
<th>ID</th>
<th>Major</th>
<th>GPA</th>
</tr>
</thead>
<tbody>
<tr>
<td>Alice</td>
<td>334322</td>
<td>CS</td>
<td>3.45</td>
</tr>
<tr>
<td>Bob</td>
<td>546346</td>
<td>Math</td>
<td>3.23</td>
</tr>
<tr>
<td>Charlie</td>
<td>045628</td>
<td>CS</td>
<td>2.75</td>
</tr>
<tr>
<td>Denise</td>
<td>964389</td>
<td>Art</td>
<td>4.0</td>
</tr>
</tbody>
</table>
ANSI SQL aliasing

- **AS keyword examples:**

  - `SELECT ID AS Student_ID
    FROM Students;`

  - `SELECT S.Major
    FROM Students AS S
    WHERE S.Name = 'Alice';`

  - `SELECT *
    FROM STUDENT AS S(I,N,S,M,G)
    WHERE S.N = 'Bob' AND S.M = 'Math';`
Oracle SQL aliasing

- **AS keyword examples:**
  - `SELECT ID AS Student_ID
    FROM Students;`
  - `SELECT S.Major
    FROM Students S
    WHERE S.Name = 'Alice';`
Aggregation

- SQL functions:
  - MIN
  - MAX
  - SUM
  - AVG
  - COUNT

- Consider Employee(Ssn, Salary, Deduction, Birthdate, Years):
  - SELECT SUM (Salary) AS TotalSalaries,
    MAX (Salary) AS MaxSalary,
    MIN (Salary) AS MinSalary,
    AVG (Salary) AS AvgSalary,
    COUNT (*) AS Cardinality,
    COUNT(DISTINCT Salary) AS SalaryLevels
  FROM Employee;
Arithmetic operators

- +, -, *, and / may be applied on numeric values in any expression
  - SELECT 1.1 * SUM(Salary) FROM Employee;

- Increment (+) and decrement (-) may be applied on date data types: date, time, and timestamp
  - SELECT Name, (CURRENT_DATE - BirthDate) AS Age FROM Employee
    WHERE (CURRENT_DATE - BirthDate) INTERVAL YEAR > 60;
Grouping example

```
SELECT Major, COUNT(*) AS NumStudents
FROM Students
WHERE GPA > 3.5
GROUP BY Major
HAVING COUNT(*) >= 5;
```

WHERE clause evaluated first, then grouping occurs

Multiple group by terms allowable
SELECT * 
FROM Students 
WHERE GPA > 3.5 
ORDER BY Name ASC, GPA DESC;
Much Ado About NULLthing

- SELECT * FROM Students WHERE NOT(Major = NULL);
- Comparison to NULL yields UNKNOWN!
- NULL values must be considered explicitly!
  - IS NULL and IS NOT NULL
- SQL provides operators to test for specific conditions
  - IS FALSE and IS NOT FALSE
  - IS TRUE and IS NOT TRUE
  - IS UNKNOWN and IS NOT UNKNOWN
● Implements if/then/else functionality

● SELECT ID, Name, CASE Major

  WHEN IS NULL THEN 'undecided'
  WHEN 'CS' THEN 'our student'
  ELSE 'busy elsewhere'

END AS Status

FROM Students
WHERE GPA > 3.0;
SELECT ID, Name, CASE
    WHEN Major = 'CS' THEN 'already here'
    WHEN Major IS NULL AND GPA > 3.5 THEN 'go after'
    WHEN Major IS NULL AND GPA > 2.75 THEN 'recruit'
    ELSE 'ignore'
END AS RecruitmentStrategy
FROM Students;
Combining data from multiple tables

Given:

- Students(ID, Name, Ssn, Major, GPA)
- Enrollment(Stud_ID, Course)
- Depts(Dept, Chair)
- Faculty(Fname, Lname, Dept)

How can we express the following relational algebra operations in SQL?

- Faculty × Depts
- Students ⋈ \_{ID = Stud_ID} Enrollment
The JOIN statement

- Introduced in SQL-92
- Specifies join condition in the `from` clause
  
  ```sql
  SELECT ID, S.Name
  FROM (Students S JOIN Enrollment E ON S.ID = E.Stud_ID)
  WHERE Course = 'CS 1555';
  ```

- Also has NATURAL JOIN:
  
  ```sql
  SELECT *
  FROM (Depts NATURAL JOIN Faculty);
  ```
Also has...

- **LEFT OUTER JOIN**
  - (or just LEFT JOIN)
- **RIGHT OUTER JOIN**
  - (or just RIGHT JOIN)
- **FULL OUTER JOIN**
  - (or just FULL JOIN)
- **CROSS JOIN**
  - Cartesian product
- **UNION JOIN**
  - Outer union
Example schema for this lecture

**EMPLOYEE**
- Fname
- Minit
- Lname
- Ssn
- Bdate
- Address
- Sex
- Salary
- Super_ssn
- Dno

**DEPARTMENT**
- Dname
- Dnumber
- Mgr_ssn
- Mgr_start_date

**DEPT_LOCATIONS**
- Dnumber
- Dlocation

**PROJECT**
- Pname
- Pnumber
- Plocation
- Dnum

**WORKS_ON**
- Essn
- Pno
- Hours

**DEPENDENT**
- Essn
- Dependent_name
- Sex
- Bdate
- Relationship
SELECT Dname || ' ' || Dnumber AS DeptID
FROM DEPARTMENT, EMPLOYEE
WHERE Fname || Lname = 'BobSmith'
   AND Mgr_ssn = Ssn;
SQL pattern matching

- **LIKE keyword**
  - Used to compare against partial strings
    - `%` in a partial string matches an arbitrary number of characters
      - Spaces included
      - ESCAPE keyword allows us to match literal `%`
    - `_` matches a single character

- **Examples:**
  - `Phone_number LIKE '412-62%'`
  - `Phone_number LIKE '___-___-1111'`
  - `Discont LIKE '1_&_%' ESCAPE '&'`
Oracle regular expression functions

- `REGEXP_LIKE(x, pattern [, match_option])`
  - True if the source x matches the regular expression pattern
  - match_option can change the default matching:
    - 'c', which specifies case sensitive matching (default)
    - 'i', which specifies case insensitive matching
    - 'n', which allows you to use the match-any-character operator
    - 'm', which treats x as multiple line
Oracle regular expression patterns

- .
- +
- ?
- *
- {m,n}
- [ ... ]
- [^ ... ]

- |
- ( ... )
- \n
- \`
- ^
- $
Oracle regular expression functions

- **REGEXP_INSTR(x, pattern [, start [, occurrence [, return_option [, match_option]]]]])**
  - Searches for pattern in x and returns the position at which pattern occurs

- **Options:**
  - **start**: position to begin the search
  - **occurrence**: indicates which occurrence of pattern_exp should be returned.
  - **return_option**: that indicates what integer to return:
    - 0 specifies the integer to return is the position of the first character in x
    - non-zero specifies the integer to return is the position of the character in x after the occurrence
  - **match_option**: same as with REGEXP_LIKE
Oracle regular expression functions

- **REGEXP_REPLACE(x, pattern [, replace_string [, start [, occurrence [, match_option]]]]])**
  - Searches x for pattern and replaces it with replace_string

- **REGEXP_SUBSTR(x, pattern [, start [, occurrence [, match_option]]]]])**
  - Returns a substring of x that matches pattern, which begins at the position specified by start!
**Range queries and conditions**

- `SELECT *`  
  
  `FROM EMPLOYEE`  
  
  `WHERE Salary >= 25000 AND Salary <= 35000;`

- Can also use a range keyword BETWEEN
  - Or its negation NOT BETWEEN
  - `SELECT *`  
    
    `FROM EMPLOYEE`  
    
    `WHERE (Salary BETWEEN 25000 AND 35000);`

- Can be use with number, character, and date data types
SQL set operations

- SQL supports UNION, EXCEPT (difference), INTERSECT
  - Not all vendors support INTERSECT, however
- UNION ALL retains duplicates
- Tables must be union-compatible!

- (SELECT ID
  FROM Students
  WHERE Major = 'CS')
  UNION
- (SELECT ID
  FROM Students
  WHERE Major = 'Math');
What a wonderful idea!

Set comparisons can be applied to two instances of sets:

- Explicit definitions within ():
  - (1, 2, 3)
  - ('CS1555', 'CS1501', 'CS441')

- Implicit definitions as a nested subquery
SELECT Fname, Lname, Ssn
FROM EMPLOYEE, DEPARTMENT
WHERE Dname NOT IN ('Marketing', 'Finance')
  AND Dno = Dnumber;
Set membership: IN

- SELECT DISTINCT Pnumber
  FROM PROJECT
  WHERE Pnumber IN (SELECT Pnumber
                       FROM PROJECT, DEPARTMENT, EMPLOYEE
                       WHERE Dnum = Dnumber AND
                       Mgr_Ssn = Ssn AND Lname = 'Smith');
Set comparisons

- Can use comparison operators on sets when quantified with ANY or ALL

- ```
SELECT M.Ssn
FROM EMPLOYEE AS M, DEPARTMENT AS D
WHERE D.Mgr_ssn = M.Ssn AND
  M.Salary < ANY (
    SELECT E.Salary
    FROM EMPLOYEE AS E
    WHERE E.Ssn NOT IN
      (SELECT DISTINCT Mgr_ssn
       FROM DEPARTMENT)
  );
```
EXISTS

- Tests if the result of a nested query is empty
- SELECT Fname, Lname
  FROM EMPLOYEE
  WHERE NOT EXISTS (SELECT *
                     FROM DEPENDENT
                     WHERE Ssn = Essn);

- SELECT Fname, Lname
  FROM EMPLOYEE
  WHERE EXISTS (SELECT *
                FROM DEPENDENT
                WHERE Ssn = Essn)
  AND EXISTS (SELECT *
               FROM DEPARTMENT
               WHERE Ssn = Mgr_ssn);
What sorcery is this??

- When a nested query references an attribute of an outer query, the queries are said to be *correlated*
  - For every tuple (or combination of tuples) from the outer query, evaluate the nested query
- Tests if the following is a set or a bag

  ```sql
  SELECT S.ID
  FROM Students S
  WHERE NOT UNIQUE ( SELECT *
      FROM (SELECT ID
        FROM Students
        WHERE Major = 'CS')
    UNION ALL
    (SELECT ID
        FROM Students
        WHERE Major = 'Math')
    WHERE S.ID = ID
  )
  ```
Scalar subqueries

- A subqueries that has an output with both a cardinality and arity of 1 (i.e., a single value)

- SELECT Fname, Lname
  FROM EMPLOYEE
  WHERE Ssn = (SELECT Mgr_ssn
                FROM DEPARTMENT
                WHERE Dname = 'Research');

- SELECT E.Ssn, (SELECT DISTINCT MAX Hours
                   FROM WORKS_ON AS W
                   WHERE E.Ssn = W.Essn) AS Hours
  FROM EMPLOYEE AS E
  WHERE Salary < 30000;
Tricky (but common) query type

- Rank employees by their salaries, be sure to consider ties

<table>
<thead>
<tr>
<th>Ssn</th>
<th>Salary</th>
<th>Rank</th>
</tr>
</thead>
<tbody>
<tr>
<td>1234567</td>
<td>100000</td>
<td>1</td>
</tr>
<tr>
<td>3456789</td>
<td>80000</td>
<td>2</td>
</tr>
<tr>
<td>9876543</td>
<td>80000</td>
<td>2</td>
</tr>
<tr>
<td>7654321</td>
<td>78000</td>
<td>4</td>
</tr>
</tbody>
</table>
Limiting result size

- [ OFFSET <start> {ROW | ROWS} ]

  FETCH {FIRST | NEXT} <count> {ROW | ROWS} ONLY

  - the OFFSET clause must come before the FETCH clause
  - <start>: number of rows to skip [default 0]
  - <count>: maximum number of rows to return [default 1]

- SELECT *

  FROM EMPLOYEE

  FETCH FIRST 10 ROWS ONLY;
Top-K queries

- **SELECT * FROM EMPLOYEE ORDER BY Salary DESC FETCH FIRST 10 ROWS ONLY;**

- **Next-K:**
  - **SELECT * FROM EMPLOYEE ORDER BY Salary DESC OFFSET 6 ROWS FETCH NEXT 10 ROWS ONLY;**
FETCH ... is the SQL standard

- Introduced in SQL:2008
- Still not widely supported
  - IBM DB2, Sybase SQL Anywhere, PostgreSQL, Microsoft SQL Server 2012, and Oracle 12c claim support
  - Other DBMSs have their own approach
- Oracle pre-12c used a *rownum* pseudocolumn
Oracle pre-12c used a *rownum* pseudocolumn

- SELECT *
  FROM (SELECT * FROM Students ORDER BY GPA DESC) AS S
  WHERE rownum <= 3
  ORDER BY rownum;

- SELECT *
  FROM (SELECT * FROM Students ORDER BY GPA DESC)
  WHERE rownum BETWEEN 3 AND 6;
We can use nested queries to insert tuples into a table

- `INSERT INTO Dept_Info (Dept_Name, Num_Students)
  SELECT Major, Count(*)
  FROM Students
  GROUP BY Major;`
Update

- Can change tuple values in a *single* relation
- `UPDATE EMPLOYEE
  SET Lname = 'Johnson'
  WHERE Ssn = '1234567';`
- `UPDATE EMPLOYEE
  SET Salary = Salary * 1.1
  WHERE Dno IN (SELECT Dnumber
                 FROM DEPARTMENT
                 WHERE Dname = 'Research');`
DELETE FROM EMPLOYEE
WHERE Ssn = '3456789';

DELETE FROM EMPLOYEE
WHERE Dno IN (SELECT Dnumber
               FROM DEPARTMENT
               WHERE Dname = 'Marketing');

DELETE FROM EMPLOYEE;