ECE 650
Systems Programming & Engineering

Spring 2018

Introduction to SQL

Tyler Bletsch
Duke University

Slides are adapted from Brian Rogers (Duke)

SQL

- **S**tructured **Q**uery **L**anguage
- Major reason for commercial success of relational DBs
  - Became a standard for relational DBs
  - Used by many database management systems (DBMS)
  - Makes it easier to move DB apps from one DBMS to another
    - If DB apps use only features that are part of the standard
  - Also lets DB apps access data stored in multiple DBMS's

Relational Algebra vs. SQL Queries

- Relational algebra written as a sequence of operations
  - Requires specifying the *order* to execute query operations
  - This is complex and restrictive for users
- SQL language provides high-level declarative language
  - User specifies only *what* the result should be
  - DBMS optimizes and decides about how to execute query

SQL Terminology

- Table = Relation
- Row = Tuple
- Column = Attribute
- Commands for data definition are
  - CREATE, ALTER, DROP
- One basic command for retrieving (querying) information
  - SELECT

Tables

- 'CREATE TABLE' command creates a new relation
  - Give table a name, specify its attributes and constraints
  - For each attribute in the table:
    - Attribute name, data type (domain of values), constraints
  - Key, entity and referential integrity constraints for the table specified after the list of attributes

- 'DROP TABLE' command removes a table

CREATE TABLE Example

```
CREATE TABLE Employee
  ( FNAME      VARCHAR(15)      NOT NULL,
    MINIT      CHAR,
    LNAME      VARCHAR(15)      NOT NULL,
    SSN        CHAR(9)          NOT NULL,
    BATE       DATE,
    ADDRESS    VARCHAR(30),
    SALARY     DECIMAL(10,2),
    SUPERSSN   CHAR(9),
    DNO        INT              NOT NULL,
  PRIMARY KEY (SSN),
  FOREIGN KEY (SUPERSSN) REFERENCES EMPLOYEE(SSN)
  FOREIGN KEY (DNO) REFERENCES DEPARTMENT(DNUMBER));
```

SQL Domains (Data Types)

- Numeric types
  - INT, SMALLINT
  - FLOAT, REAL, DOUBLE PRECISION
  - Formats: DECIMAL(i,j) (i=precision, j=scale)
- Character string
  - Fixed length: CHAR(n) or CHARACTER(n)
  - Variable length: VARCHAR(n) or CHAR VARYING(n)
    - n=max # of chars
  - Bit string: BIT(n) or BIT VARYING(n)
- Date and Time
  - DATE=YYYY-MM-DD, TIME=HH:MM:SS
  - TIMESTAMP includes both date and time
- Can also create a domain (like a typedef)
  - CREATE DOMAIN SSN_TYPE AS CHAR(9)

Default Values

- Can define a default value for an attribute
- Use DEFAULT <value> notation
  - If not specified, default is Null
- E.g.:

```
CREATE TABLE Employee
   ( FNAME      VARCHAR(15)      NOT NULL,
     <snip>
     DNO        INT              NOT NULL DEFAULT 1,
   PRIMARY KEY (SSN),
   FOREIGN KEY (SUPERSSN) REFERENCES EMPLOYEE(SSN)
   FOREIGN KEY (DNO) REFERENCES DEPARTMENT(DNUMBER));
```

Referential Integrity Actions

- What should happen if referential integrity is violated
  - Recall this can happen as tuples are inserted or deleted
- Can specify a *referential triggered action* on foreign key
  - Options are:
    - SET NULL – Set foreign key attribute to NULL
    - CASCADE – Set foreign key attribute to updated value
    - SET DEFAULT – Set foreign key to default value
  - Must be qualified with one of:
    - ON DELETE – If tuple referenced by foreign key is deleted
    - ON UPDATE – If tuple referenced by foreign key is updated

Referential Integrity Actions (2)

- Examples
  - SET NULL ON DELETE: If tuple referenced by a foreign key is deleted, set the foreign key field to NULL in referencing tuples
  - CASCADE ON UPDATE: If tuple referenced by a foreign key is updated, update the foreign key value in referencing tuples
  - CASCADE ON DELETE: If a tuple referenced by a foreign key is deleted, delete referencing tuples
- Can also give a constraint a name (optional)

```
CREATE TABLE Employee
  ( <snip>
  CONSTRAINT EMPPK PRIMARY KEY (SSN),
  CONSTRAINT EMPSUPERFK FOREIGN KEY (SUPERSSN) REFERENCES
EMPLOYEE(SSN) ON DELETE SET NULL ON UPDATE CASCADE,
  <snip>);
```

Modify a Table

- ALTER TABLE
  - Add or drop columns (attributes)
  - Change column definitions
  - Add or drop table constraints
- Add attribute to a table:
  - ALTER TABLE Employee ADD JOB VARCHAR(12);
- Drop attribute from a table
  - ALTER TABLE Employee DROP ADRESS CASCADE;
    - Must choose either CASCADE or RESTRICT
    - CASCADE: constraints referencing this column are also dropped
    - RESTRICT: operation only succeeds if no constraints refer to column

Basic Queries

- SELECT statement
  - For retrieving database information
- Distinction between SQL and formal relational model
  - SQL allows a table to have 2 or more tuples identical in all values
  - SQL table is thus not a *set* of tuples
    - It is a *multiset*
  - Some SQL relations are constrained to be sets
    - Due to key constraint
  - Something to be aware of as we discuss queries

Example Relational Database Tables

- COMPANY = {EMPLOYEE, DEPARTMENT, DEPT_LOCATIONS, PROJECT, WORKS_ON, DEPENDENT}

EMPLOYEE

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO

DEPARTMENT

DNAME	DNUMBER	MGRSSN	MGRSTARTDATE

DEPT LOCATIONS

DNUMBER	DLOCATION

PROJECT

PNAME	PNUMBER	PLOCATION	DNUM

WORKS_ON

ESSN	PNO	HOURS

DEPENDENT

ESSN	DEP_NAME	SEX	BDATE	RELATIONSHIP

SELECT-FROM-WHERE

- Basic SELECT statement form:
  - SELECT <attribute list> // list of attribute names to return
  - FROM <table list> // list of table names to process the query
  - WHERE <condition>; // conditional expression to identify tuples

- Example:
  - SELECT BDATE, ADDRESS FROM EMPLOYEE WHERE FNAME='John' AND MINIT='B' AND LNAME='Smith';
  - Similar to the relational algebra expression:
    - $\pi_{BDATE,ADDRESS}(\sigma_{FNAME='John' \text{ AND } MINIT='B' \text{ AND } LNAME='Smith'}(EMPLOYEE))$
  - SELECT-clause specifies projection attributes
  - WHERE-clause specifies selection condition

Multiple Tables

- SELECT FNAME, LNAME, ADDRESS

  FROM EMPLOYEE, DEPARTMENT

  WHERE DNAME='Research' AND DNUMBER=DNO
  - Like a SELECT-PROJECT-JOIN sequence of relational algebra ops
  - DNAME='Research' is a *selection condition*
  - DNUMBER=DNO is a *join condition*
- SELECT PNUMBER, DNUM, LNAME, ADDRESS, BDATE

  FROM PROJECT, DEPARTMENT, EMPLOYEE

  WHERE DNUM=DNUMBER AND MGRSSN=SSN AND
  PLOCATION='Stafford'
  - Two join conditions here
  - DNUM=DNUMBER relates a project to its controlling department
  - MGRSSN=SSN relates the controlling department to the employee managing it

Dealing with Ambiguous Attribute Names

- Same name may be used by different attributes in different tables (relations)
- In that case, must qualify the attribute name with relation name
  - Prefix relation name to attribute name
  - Separate two by a period
- For example, if both EMPLOYEE and DEPARTMENT tables used fields named NAME and DNUMBER (instead of DNAME and DNO)
- SELECT FNAME, LNAME, ADDRESS

  FROM EMPLOYEE, DEPARTMENT

  WHERE DEPARTMENT.NAME='Research' AND
  DEPARTMENT.DNUMBER=EMPLOYEE.DNUMBER

Aliasing

- Can declare alternative relation names
  - And even attribute names for the relation
- SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME

  FROM EMPLOYEE AS E, DEPARTMENT AS S

  WHERE E.SUPERSSN=S.SSN;
- Think of E and S as two copies of same table
  - Allows us to join the two copies of the same table
  - Shows manager name for each employee name
- Can also alias the attribute names
  - EMPLOYEE AS E(FN, MI, LN, SSN, BD, ADDR, SEX, SAL, SSSN, DNO)

Unspecified WHERE-Clause

- SELECT SSN FROM EMPLOYEE;
  - Select all EMPLOYEE SSNs
- SELECT SSN, DNAME FROM EMPLOYEE, DEPARTMENT;
  - Select all combinations of EMPLOYEE SSN and DEPARTMENT DNAME
- Important to specify every selection and join condition in the WHERE Clause
  - Otherwise may end up w/ very large result relations (cross product)

Retrieving All Attributes

- What if we want all attributes of a high-degree table?
  - Do not need to list them all in SELECT Clause
  - Can use the asterisk (*)
- SELECT * FROM EMPLOYEE WHERE DNO=5;
  - Retrieve all attributes of EMPLOYEE tuples who work in department number 5
- SELECT * FROM EMPLOYEE, DEPARTMENT WHERE DNAME='Research' AND DNO=DNUMBER
  - Retrieve all attributes of an EMPLOYEE and all attributes of their DEPARTMENT for every employee of 'Research' department

LIKE clause

- Allows comparison conditions on parts of a string
  - Two special characters:
    - '%' replaces an arbitrary number of characters
    - '_' replaces a single character
- SELECT FNAME, LNAME

  FROM EMPLOYEE

  WHERE ADDRESS LIKE '%Houston,TX%';
  - Retrieve all employees whose address is in Houston, Texas
- SELECT FNAME, LNAME

  FROM EMPLOYEE

  WHERE BDATE LIKE '_ _ 5 _ _ _ _ _ _ _ _';
  - Retrieve all employees who were born during the 1950s
  - Where BDATE format is 'YYYY-MM-DD'

Arithmetic Operators

- We can use arithmetic on numeric domains
  - add, subtract, multiply, divide
- SELECT FNAME, LNAME, 1.1*SALARY

  FROM EMPLOYEE, WORKS_ON, PROJECT

  WHERE SSN=ESSN AND PNO=PNUMBER AND

  PNAME='ProductX';
  - Want to see effect of giving all employees who work on ProductX a 10% raise

Other Operators

- Can append strings with concatenate operator: '||'
  - But that's logical OR in the rest of the world other than databases, so:
    - Some SQL implementations use + operator
    - Some SQL implementations use a CONCAT function
- Increment and decrement operators for
  - Date, time, timestamp, interval data types
- BETWEEN operator (for convenience):
- SELECT *

  FROM EMPLOYEE

  WHERE (SALARY **BETWEEN** 30000 AND 40000) AND DNO=5;
  - Retrieve all employees in dept. 5 whose salary is between $30,000 and $40,000

ORDER BY Clause

- Sometimes desirable to re-order returned results
- Can use ORDER BY clause
- SELECT DNAME, LNAME, FNAME, PNAME

  FROM DEPARTMENT, EMPLOYEE, WORKS_ON, PROJECT

  WHERE DNUMBER=DNO AND SSN=ESSN AND

  PNO=PNUMBER

  ORDER BY DNAME, LNAME, FNAME
  - Retrieve a list of employees and projects they are working on
  - Ordered by department, and within each department, ordered alphabetically by last name, first name

ORDER BY Clause (2)

- Can also specify ascending or descending order
  - ASC or DESC keyword
- Example:
  - **ORDER BY** DNAME **DESC**, LNAME **ASC**, FNAME **ASC**

Nested Queries

- Some queries require fetching existing DB values and using them in a comparison condition
- Useful to use nested queries
  - SELECT, FROM, WHERE blocks inside WHERE of other query
  - Other query is called **outer query**

Example Relational Database Tables

- COMPANY = {EMPLOYEE, DEPARTMENT, DEPT_LOCATIONS, PROJECT, WORKS_ON, DEPENDENT}

EMPLOYEE

FNAME	MINIT	LNAME	SSN	BDATE	ADDRESS	SEX	SALARY	SUPERSSN	DNO

DEPARTMENT

DNAME	DNUMBER	MGRSSN	MGRSTARTDATE

DEPT LOCATIONS

DNUMBER	DLOCATION

PROJECT

PNAME	PNUMBER	PLOCATION	DNUM

WORKS_ON

ESSN	PNO	HOURS

DEPENDENT

ESSN	DEP_NAME	SEX	BDATE	RELATIONSHIP

Nested Query Example

- SELECT **DISTINCT** PNUMBER

  FROM PROJECT

  WHERE PNUMBER **IN** (SELECT PNUMBER

  FROM PROJECT, DEPARTMENT,

  EMPLOYEE

  WHERE DNUM=DNUMBER AND

  MGRSSN=SSN AND LNAME='Smith')

  OR

  PNUMBER **IN** (SELECT PNO FROM WORKS_ON, EMPLOYE

  WHERE ESSN=SSN AND LNAME='Smith');

Select project numbers of projects with 'Smith' involved as a manager

Select project numbers of projects with 'Smith' involved as a worker

IN compares a value v with a set; evaluates to true if v is aan element in the set

More on IN Operator

- Can compare tuple of values in parenthesis with a set of union-compatible tuples
- SELECT DISTINCT ESSN

  FROM WORKS_ON

  WHERE (PNO, HOURS) IN (SELECT PNO, HOURS

  FROM WORKS_ON

  WHERE ESSN='123456789');
  - Select SSN of employees working the same (project, hours) combination on some project that employee with SSN 123456789 works on

ANY, SOME, ALL Keywords

- ANY and SOME operators have same meaning
  - Can use equivalently to IN
  - E.g. WHERE PNUMBER = ANY …
    - instead of WHERE PNUMBER IN …
  - Can also combine with operators for comparison (>, >=, <, <=)
- ALL
  - Compares a value ‘v’ to every value in a set
  - SELECT LNAME, FNAME
    FROM EMPLOYEE
    WHERE SALARY > ALL (SELECT SALARY FROM EMPLOYEE
    WHERE DNO=5);
    Returns names of employees whose salary is greater than salary of all employees in department 5

Correlated Nested Queries

- Correlated condition:
  - When condition in WHERE-clause of a nested query refers to some attribute of a relation declared in the outer query
- Consider that the nested query is evaluated once for each tuple in the outer query
- For example –
- SELECT E.FNAME, E.LNAME

  FROM EMPLOYEE AS E

  WHERE E.SSN IN (SELECT ESSN FROM DEPENDENT

  WHERE E.FNAME=DEPENDENT_NAME);

Correlated Nested Queries (2)

- In general:
  - For query written with nested SELECT, FROM, WHERE blocks
  - And using the = or IN operators
  - Can always be expressed as a single query block

- For example, can rewrite query from previous slide as:
- SELECT E.FNAME, E.LNAME

  FROM EMPLOYEE AS E, DEPENDENT AS D

  WHERE E.SSN=D.ESSN AND

  E.FNAME=D.DEPENDENT_NAME;

EXISTS Function

- Check whether result of correlated nested query is empty
  - Empty means contains no tuples
- SELECT E.FNAME, E.LNAME

  FROM EMPLOYEE AS E

  WHERE **EXISTS** IN (SELECT * FROM DEPENDENT

  WHERE E.SSN=ESSN AND

  E.FNAME= DEPENDENT_NAME);
- Can also use “NOT EXISTS”
- SELECT FNAME, LNAME

  FROM EMPLOYEE

  WHERE **NOT EXISTS** (SELECT * FROM DEPENDENT

  WHERE SSN=ESSN);

Find names of employees who have no dependents

UNIQUE Function

- UNIQUE(Q)
  - Returns true if there are no duplicate tuples in the query Q
  - Otherwise returns false

Explicit Sets and NULLs

- WHERE-clause may contain explicit set of values
  - Enclosed in parenthesis
- Example:
  - SELECT DISTINCT ESSN
    FROM WORKS_ON
    WHERE PNO IN (1, 2, 3);

All employee SSNs who work on projects 1, 2, or 3

- SQL allows queries to check whether a value is NULL
  - NULL means missing or undefined or not applicable
  - Must use “IS” or “IS NOT” instead of = or ≠
  - SELECT FNAME, LNAME
    FROM EMPLOYEE
    WHERE SUPERSSN IS NULL;

All employees who do not have supervisors

Joined Table

- Specify a table resulting from a join operation
  - In the FROM-clause of a query
  - May be easier to follow than mixing together all the select and join conditions in the WHERE-clause
- Example:
  - Retrieve name and address of every employee who works for the ‘Research’ department
  - SELECT FNAME, LNAME, ADDRESS

    FROM (EMPLOYEE **JOIN** DEPARTMENT **ON** DNO=DNUMBER) WHERE DNAME='Research';
- Can also use ‘NATURAL JOIN’:
  - No join condition is specified (e.g. ‘ON’ clause)

Aggregate Functions

- Built-in functions:
  - COUNT, SUM, MIN, MAX, AVG
  - COUNT: # of tuples or values specified in a query
- Find sum of salaries of all employees of the 'Research' department, as well as max, min, & average salaries
  - SELECT SUM(SALARY), MAX(SALARY), MIN(SALARY), AVG(SALARY)
    FROM EMPLOYEE, DEPARTMENT
    WHERE DNO=DNUMBER AND DNAME='Research';
- Retrieve the number of employees in the company
  - SELECT COUNT(*) FROM EMPLOYEE;
- Count the # of distinct salary values in the database
  - SELECT COUNT(DISTINCT SALARY) FROM EMPLOYEE;

GROUP BY Clause

- Sometimes want to apply aggregate functions to subgroups of tuples in a relation
  - E.g. find average salary of employees in each department
  - GROUP BY clause specifies the grouping attributes which should also appear in the SELECT-clause
- Example: for each department, retrieve the department number, number of employees in dept., and avg salary
  - SELECT DNO, COUNT(*), AVG(SALARY)
    FROM EMPLOYEE
    GROUP BY DNO;
- Example: retrieve the project number, project name, and the # of employees who work on that project
  - SELECT PNUMBER, PNAME, COUNT(*)
    FROM PROJECT, WORKS_ON
    WHERE PNUMBER=PNO
    GROUP BY PNUMBER, PNAME;

SQL Views

- Views (also called Virtual Tables)
  - Single table derived from other tables
  - Does not necessarily exist in physical form (e.g. stored in dbase)
  - Can think of as way to specify a table we need to reference often
    - E.g. instead of JOIN on several tables every time for certain query
- Example:
  - CREATE VIEW WORKS_ON1

    AS SELECT FNAME, LNAME, PNAME, HOURS

    FROM EMPLOYEE, PROJECT, WORKS_ON

    WHERE SSN=ESSN AND PNO=PNUMBER;
  - Creates view with first name, last name, project name, and hours for each employee's project

Using SQL safely

- What if your app wants to allow a user to search the database?

```
String sql = "SELECT ... FROM persons WHERE name = '" + userinput + "'";
DbCommand cmd = new DbCommand(connection, sql);
Result res = cmd.Execute();
String name = res[0]["FIRST_NAME"];
```

- What if the user input is

```
'; INSERT INTO USERS VALUES ('hacker', 'mypassword', True);
```

- You got hacked!!

- Mediocre solution: *escape* all user input (e.g., replace ' with \')
- Better solution: use *prepared statements* (database library lets you create queries with placeholders that get filled with variables.
- Best solution: Don't make SQL queries yourself – use an ORM (next slide)

SQL injection example

Object Relational Mapping

- **Object Relational Mapping (ORM)**: Library layer that connects classes/objects to entities in the database
  - Removes need to construct queries in most cases
  - *Is usually a good idea!!*
- Without ORM:

```
String sql = "SELECT ... FROM persons WHERE id = 10";
DbCommand cmd = new DbCommand(connection, sql);
Result res = cmd.Execute();
String name = res[0]["FIRST_NAME"];
```

- With ORM:

```
Person p = repository.GetPerson(10);
String name = p.getFirstName();
```

- Also avoids the risk of SQL injection!