# CS354: Database

# SQL Outline

- Data definition
  - Database Creation
  - Table Creation
- Query (SELECT)
- Data update (INSERT, DELETE, UPDATE)
- View definition

# Basic SQL Retrieval Query

- Basic form of SELECT statement is called a *mapping* or a *SELECT-FROM-WHERE block*

**SELECT <attribute list>**
**FROM <table list>**
**WHERE <condition on the tables>**

cartesian product of relations is formed

conditions of the form attr1 op constant/attr2

- Translated to relational algebra expression

$$\pi_{<\text{attribute list}>}\sigma_{<\text{condition}>}(R_1 \times R_2 \times \cdots \times R_n)$$

- Does not remove duplicates as SELECT in relational algebra

# Example Query: Basic

- Retrieve the birthdate and address of the employee whose name is 'John B. Smith"

```
SELECT bdate, address
FROM   employee
WHERE  fname='John' AND minit='B'
AND    lname='Smith';
```

- Similar to SELECT-PROJECT pair of relational algebra operations

- Result may contain duplicate tuples

# Example Query (1)

List the SSN, last name, and department number of all employees

```
SELECT ssn, lname, dno
FROM   employee;
```

unspecified WHERE clause indicates no condition

| ssn | lname | dno |
|---|---|---|
| 111-12-2345 | Kirk | 5 |
| 222-23-2222 | McCoy | 4 |
| 134-23-2345 | Sulu | 4 |
| 234-13-3840 | Chapel | 1 |
| 134-52-2340 | Scott | 5 |

# Example Query (2)

List the department number and name of all departments

```
SELECT dnumber, dname
FROM   department;
```

| dnumber | dname |
|---|---|
| 5 | Research |
| 4 | Administration |
| 1 | Headquarters |

# Example Query (3)

```
SELECT ssn, lname, dno, dnumber, dname
FROM   employee, department;
```

| ssn | lname | dno | dnumber | dname |
|---|---|---|---|---|
| 111-12-2345 | Kirk | 5 | 5 | Research |
| 111-12-2345 | Kirk | 5 | 4 | Administration |
| 111-12-2345 | Kirk | 5 | 1 | Headquarters |
| 222-23-2222 | McCoy | 4 | 5 | Research |
| 222-23-2222 | McCoy | 4 | 4 | Administration |
| 222-23-2222 | McCoy | 4 | 1 | Headquarters |
| … | … | … | … | … |
| 134-52-2340 | Scott | 5 | 5 | Research |
| 134-52-2340 | Scott | 5 | 4 | Administration |
| 134-52-2340 | Scott | 5 | 1 | Headquarters |

# SQL: Join Operation

- Relational algebra expression

$$R_1 \bowtie_{\text{condition}} R_2 = \sigma_{\text{condition}}(R_1 \times R_2)$$

  - Cartesian product followed by a selection operation

- SQL command

  - **FROM** clause specifies Cartesian product operation

  - **WHERE** clause specifies condition of the selection operation

# Example Query: Join

```
SELECT ssn, lname, dno, dnumber, dname
FROM   employee, department
WHERE  dno = dnumber;
```

| ssn | lname | dno | dnumber | dname |
|---|---|---|---|---|
| 111-12-2345 | Kirk | 5 | 5 | Research |
| 222-23-2222 | McCoy | 4 | 4 | Administration |
| 134-23-2345 | Sulu | 4 | 4 | Administration |
| 234-13-3840 | Chapel | 1 | 1 | Headquarters |
| 134-52-2340 | Scott | 5 | 5 | Research |

# Example Query (4)

Find the name and address of employees working in the 'Research' department

- Relational Algebra expression

$$\mathrm{RD} = \sigma_{\mathrm{Dname}=\text{'Research'}}(\mathrm{DEPARTMENT})$$

$$\mathrm{RE} = \mathrm{RD} \bowtie_{\mathrm{Dnumber} = \mathrm{Dno}} \mathrm{EMPLOYEE}$$

$$\mathrm{Answer} = \pi_{\mathrm{fname,lname,Address}}(\mathrm{RE})$$

- SQL expression

```
SELECT ssn, lname, dno, dnumber, dname
FROM   employee, department
WHERE dname='Research' AND dno = dnumber;
```

selection condition  join condition

# Example Query (5)

Find the name of employees in the ‘Research’ department who earn over $30,000

# Example Query (6)

Find the SSN of employees who work on the project ‘ProductX’

# Example Query (7)

Find the name of employees who work on the project ‘ProductX’

# Example Query (8)

For the projects located in ‘Stafford’, find the name of the project, the name of the controlling department, the last name of the department’s manager, his address, and birthdate

# SQL: DISTINCT

- SQL outputs duplicate values by default
  - Each relation is a multi-set (bag) of tuples as opposed to a set of tuples
  - Favored for database efficiency
- **DISTINCT** keyword in **SELECT** clause removes duplicate values
- Downside: requires sorting of the tuples (heavy duty processing)

# Example Query: DISTINCT

- SELECT name
  FROM dependent;

| name |
| --- |
| James |
| Spock |
| Uhura |
| James |
| Hikaru |

- SELECT DISTINCT name
  FROM dependent;

| name |
| --- |
| James |
| Spock |
| Uhura |
| Hikaru |

# SQL: * SELECTOR

- Selects all the values of the selected tuples for *all the attributes*

- Example:

```
SELECT *
FROM   department
WHERE  dname = 'Research';
```

| dnumber | dname | mgrSsn | mgrStartDate |
|---|---|---|---|
| 5 | Research | 333-44-5555 | 1978-05-10 |

# SQL: Qualifying Attribute Names

- Ambiguous attribute names: the same name for two (or more) attributes in *different* relations
  Example:
  Project(essn, pno, hours);
  Dependent(essn, name, sex, bdate, relationship)

- Ambiguous attributed names that appear in the same query need to be made explicit (otherwise cannot tell which relation it is from)

- Qualify (prefix) the attribute name with the source relation name

# Example Query: Qualifying Attribute Names

Find project numbers of projects worked on by employees who have a daughter named ‘Alice’

```
SELECT   pno
FROM     works_on, dependent
WHERE    works_on.essn = dependent.essn
   AND   name = ‘Alice’;
```

# SQL: Aliasing

- Sometimes, there is a need to use the same relation multiple times in a **SELECT** command
  Example: List each employees first name, last name, and their manager's first name and last name

- Every attribute name of that relation will be ambiguous

- Use an alias or identifier that follows a relation name in the **FROM** clause of a **SELECT** command

  - No comma between alias and relation name!

  - Refer to the relation using the given alias in other parts of query

# Example Query: Aliasing

List each employees first name, last name, and their manager's first name and last name

```
SELECT e.fname, e.lname, m.fname, m.lname
FROM   employee e, employee m
WHERE  e.superssn = m.ssn;
```

e and m are called aliases or tuple variables for employee relation

# SQL: Arithmetic Operations

- Any arithmetic expression (that makes sense) can be used in the **SELECT** clause

- Example: Show the effect of giving all employees who work on the 'ProductX' project a 10% raise

```
SELECT   fname, lname, 1.1*salary
FROM     employee, works_on, project
WHERE    ssn = essn
  AND    pno = pnumber
  AND    pname = 'ProductX'
```

# SQL: Set Operations

Not all set operations have been incorporated into SQL

- **UNION**: in most implementations because it's very easy to merge 2 result sets (O(n) running time)
- **INTERSECT**: in few implementations because it's hard to intersect 2 sets (O(N log N) running time)
- **MINUS**: almost no implement provides this (just as expensive as **INTERSECT**)
- **CARTESIAN PRODUCT**: built into SELECT command

# SQL: Set Operations (2)

- Resulting relations of set operations are sets of tuples —> duplicate tuples are eliminated from the result
- Set operations apply only to union compatible relations: two relations must have the same attributes and attributes must be in the same order
- Set division is not part of the SQL standard
- MySQL only implements the UNION operator

# Example Query: UNION

Find the name of projects that are worked on by ‘Smith’ or ‘Borg’

```
(SELECT pname
 FROM    project, works_on, employee
 WHERE pnumber = pno
      AND essn = ssn
      AND lname = ‘Smith’)
UNION
(SELECT pname
 FROM    project, works_on, employee
 WHERE pnumber = pno
      AND essn = ssn
      AND lname = ‘Borg’)
```

# Example Query: UNION (2)

List all project names that involve an employee whose last name is ‘Smith” either as a worker or manager of the department that controls the project

```
(SELECT pname
 FROM    project, department, employee
 WHERE dnum = dnumber AND mgrssn = ssn
      AND lname = ‘Smith’)
UNION
(SELECT pname
 FROM    project, works_on, employee
 WHERE pnumber = pno AND essn = ssn
     AND lname = ‘Smith’)
```

# SQL: What can be used in WHERE?

- Attribute names of the relation(s) used in the **FROM** clause
- Comparison operators: =, <>, <, >, <=, >=
- Arithmetic operations: +, -, *, /
- Logical operators to combine conditions: **AND**, **OR**, **NOT**
- Operations on strings (e.g., concatenation)
- Membership test
- Pattern matching

# SQL: IN Operator

- Tests whether a value is contained in a set
  - True if attribute value is a member of the set of values
  - False otherwise
- Syntax:
  **attr IN ( set of values)**

# Example Query: IN

- Find the name of employees whose SSN is 123456789 or 333445555

```
SELECT fname, lname
FROM   employee
WHERE  ssn IN ('123456789', '333445555');
```

- Find the name of employees whose DNO is 4 or 5 and are male

```
SELECT fname, lname
FROM   employee
WHERE  (dno, sex) IN ((4, 'M'), (5, 'M'));
```

# SQL: LIKE Operator

- Substring comparison for partial strings (wildcard string comparison)
- Special wildcard characters:
  - Underscore (_) matches exactly one character (equivalent to ? in the UNIX shell)
  - Percent (%) matches 0 or more characters (equivalent to * in the UNIX shell)
  - Only used with the **LIKE** operator

# Example Query: LIKE

- Find names of employees whose last name start with 'S'

```
SELECT fname, lname
FROM   employee
WHERE  lname LIKE 'S%';
```

- Find the names of employees who live in Houston

```
SELECT fname, lname
FROM   employee
WHERE  address LIKE '%Houston%;
```

# SQL: IS NULL

- Test if an attribute contains the NULL value
- Syntax:
  **attr IS NULL**
- Example: Find employees that have NULL value in the salary attribute

```
SELECT *
FROM   employee
WHERE  salary IS NULL
```

# SQL: NOT IN and IS NOT NULL

- Tests whether a value is not contained in a set or not a null value respectively
- Syntax looks similar to the IN and IS NULL operators:
  **attr NOT IN ( set of values)**
  **attr IS NOT NULL**

# SQL: Three-Value Logic

AND

| | TRUE | FALSE | UNKNOWN |
|---|---|---|---|
| TRUE | TRUE | FALSE | UNKNOWN |
| FALSE | FALSE | FALSE | FALSE |
| UNKNOWN | UNKNOWN | FALSE | UNKNOWN |

OR

| | TRUE | FALSE | UNKNOWN |
|---|---|---|---|
| TRUE | TRUE | TRUE | TRUE |
| FALSE | TRUE | FALSE | UNKNOWN |
| UNKNOWN | TRUE | UNKNOWN | UNKNOWN |

NOT

| TRUE | FALSE | UNKNOWN |
|---|---|---|
| FALSE | TRUE | UNKNOWN |

# SQL: ORDER BY

- Sort the tuples in a query based on the values of some attributes
- Default order is in ascending order of the values (ASC)
  - Specify descending order using keyword DESC
- Syntax:

```
SELECT <attribute list>
FROM   <table list>
WHERE  <condition on the tables>
ORDER BY <attribute-list> ASC | DESC;
```

sorting by multiple columns is just separated with a comma

# Example Query: ORDER BY

- Sort employees by their salary value in descending order

```
SELECT    fname, lname, salary
FROM      employee
ORDER BY  salary DESC;
```

- Sort employees by their salary figures and within the same salary figure, by their last name

```
SELECT    fname, lname, salary
FROM      employee
ORDER BY  salary, lname;
```

# SQL: LIMIT

- Limit the output to be only the specified number of tuples
  - Useful if your table has many relations and you just want to sanity check your work
  - Can be used with ORDER BY to get a maximum or minimum value
- Syntax:

```
SELECT <attribute list>
FROM   <table list>
WHERE  <condition on the tables>
LIMIT  <number of tuples>;
```

# MySQL Workbench

- Open source, integrated development environment for MySQL database system
  - SQL Editor
  - Data modeling
  - Data administration + performance monitoring
- Works on Windows, Linux, Mac OS X
- https://www.mysql.com/products/workbench/

# SQL Queries: Recap

- Basic Query Form
  - Qualifying and Aliasing
  - * SELECTOR
  - UNION
  - DISTINCT
  - IN and LIKE
  - ORDER BY
  - LIMIT