Introduction to SQL: Data Retrieving

Ruslan Fomkin

Databasdesign för Ingenjörer – 1056F
Structured Query Language (SQL)

- History:
  - SEQUEL (Structured English QUery Language), earlier 70’s, IBM Research
  - SQL (ANSI 1986), SQL1 or SQL-86
  - SQL2 (SQL-92)
  - SQL-99 (SQL3)
    - core specification and optional specialized packages
- Standard language for commercial DBMS
  - each DBMS has own features over standard
SQL includes

- Data Definition Language (DDL)
- Data Manipulation Language (DML)
  - Queries
  - Updates
- Additional facilities
  - views
  - security and authorization
  - integrity constraints
  - transaction controls
  - rules for embedding SQL statements into, e.g., Java, C++
SQL based on

- **Formal Relational Data Model**
  - **Terminology**
    - relation - table
    - tuple - row
    - attribute - column
  - SQL allows a table to have duplicates

- **Tuple Relational Calculus**
  - Includes some operations from relational algebra
Basic query statement of SQL

```
SELECT A_1, A_2, ..., A_n
FROM r_1, r_2, ..., r_m
WHERE P
```

- $A_1, A_2, ..., A_n$ – list of the attribute names whose values to be retrieved by the query
- $r_1, r_2, ..., r_m$ – list of the table names required to process the query
- $P$ – conditional expression that identifies the tuples to be retrieved by the query
  - connectors: **AND**, **OR**, **NOT**
  - comparison operations: $=$, $<$, $<=$, $>$, $>=$, $<$>

- Result of the query is a table
Example database (from E/N ch. 5)
<table>
<thead>
<tr>
<th>EMPLOYEE</th>
<th>FNAME</th>
<th>MINIT</th>
<th>LNAME</th>
<th>SSN</th>
<th>BDATE</th>
<th>ADDRESS</th>
<th>SEX</th>
<th>SALARY</th>
<th>SUPERSSN</th>
<th>DNO</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>B</td>
<td>Smith</td>
<td></td>
<td>123456789</td>
<td>1965-01-09</td>
<td>731 Fondren, Houston, TX</td>
<td>M</td>
<td>30000</td>
<td>333445555</td>
<td>5</td>
</tr>
<tr>
<td>Franklin</td>
<td>T</td>
<td>Wong</td>
<td></td>
<td>333445555</td>
<td>1955-12-08</td>
<td>638 Vose, Houston, TX</td>
<td>M</td>
<td>40000</td>
<td>886665555</td>
<td>5</td>
</tr>
<tr>
<td>Alicia</td>
<td>J</td>
<td>Zelaya</td>
<td></td>
<td>999887777</td>
<td>1969-01-19</td>
<td>3231 Castle, Spring, TX</td>
<td>F</td>
<td>25000</td>
<td>987654321</td>
<td>4</td>
</tr>
<tr>
<td>Jennifer</td>
<td>S</td>
<td>Wallace</td>
<td></td>
<td>887654321</td>
<td>1941-05-20</td>
<td>291 Berry, Bellaire, TX</td>
<td>F</td>
<td>43000</td>
<td>888665555</td>
<td>4</td>
</tr>
<tr>
<td>Ramesh</td>
<td>K</td>
<td>Narayan</td>
<td></td>
<td>666864444</td>
<td>1962-09-15</td>
<td>975 FireOak, Humble, TX</td>
<td>M</td>
<td>38000</td>
<td>333445555</td>
<td>5</td>
</tr>
<tr>
<td>Joyce</td>
<td>A</td>
<td>English</td>
<td></td>
<td>453435453</td>
<td>1972-07-31</td>
<td>5631 Rios, Houston, TX</td>
<td>M</td>
<td>25000</td>
<td>333445555</td>
<td>5</td>
</tr>
<tr>
<td>Ahmad</td>
<td>V</td>
<td>Jaffer</td>
<td></td>
<td>587697987</td>
<td>1969-03-29</td>
<td>980 Dallas, Houston, TX</td>
<td>M</td>
<td>25000</td>
<td>987654321</td>
<td>4</td>
</tr>
<tr>
<td>James</td>
<td>E</td>
<td>Burg</td>
<td></td>
<td>888665555</td>
<td>1937-11-10</td>
<td>450 Stone, Houston, TX</td>
<td>M</td>
<td>55000</td>
<td>null</td>
<td>1</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DEPT_LOCATIONS</th>
<th>DNUMBER</th>
<th>LOCATION</th>
</tr>
</thead>
<tbody>
<tr>
<td>Research</td>
<td>5</td>
<td>Bellaire</td>
</tr>
<tr>
<td>Administration</td>
<td>4</td>
<td>Houston</td>
</tr>
<tr>
<td>Headquarters</td>
<td>1</td>
<td>Staff</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>WORKS_ON</th>
<th>ESSN</th>
<th>PNO</th>
<th>HOURS</th>
</tr>
</thead>
<tbody>
<tr>
<td>123456789</td>
<td>1</td>
<td>32.5</td>
<td></td>
</tr>
<tr>
<td>12346789</td>
<td>2</td>
<td>7.5</td>
<td></td>
</tr>
<tr>
<td>666864444</td>
<td>3</td>
<td>4.0</td>
<td></td>
</tr>
<tr>
<td>453435453</td>
<td>1</td>
<td>20.0</td>
<td></td>
</tr>
<tr>
<td>453435453</td>
<td>2</td>
<td>20.0</td>
<td></td>
</tr>
<tr>
<td>333445666</td>
<td>2</td>
<td>10.0</td>
<td></td>
</tr>
<tr>
<td>333445666</td>
<td>3</td>
<td>10.0</td>
<td></td>
</tr>
<tr>
<td>333445666</td>
<td>10</td>
<td>10.0</td>
<td></td>
</tr>
<tr>
<td>333445666</td>
<td>20</td>
<td>10.0</td>
<td></td>
</tr>
<tr>
<td>333445666</td>
<td>30</td>
<td>30.0</td>
<td></td>
</tr>
<tr>
<td>333445666</td>
<td>10</td>
<td>10.0</td>
<td></td>
</tr>
<tr>
<td>333445666</td>
<td>10</td>
<td>20.0</td>
<td></td>
</tr>
<tr>
<td>333445666</td>
<td>50</td>
<td>5.0</td>
<td></td>
</tr>
<tr>
<td>333445666</td>
<td>50</td>
<td>20.0</td>
<td></td>
</tr>
<tr>
<td>333445666</td>
<td>20</td>
<td>15.0</td>
<td></td>
</tr>
<tr>
<td>333445666</td>
<td>20</td>
<td>null</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>PROJECT</th>
<th>PNAME</th>
<th>PNUMBER</th>
<th>LOCATION</th>
<th>DNUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>ProductX</td>
<td>1</td>
<td>Bellaire</td>
<td>1</td>
<td>5</td>
</tr>
<tr>
<td>ProductY</td>
<td>2</td>
<td>Sugarland</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>ProductZ</td>
<td>3</td>
<td>Houston</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>Computerization</td>
<td>10</td>
<td>Stafford</td>
<td>10</td>
<td>4</td>
</tr>
<tr>
<td>Reorganization</td>
<td>20</td>
<td>Houston</td>
<td>20</td>
<td>1</td>
</tr>
<tr>
<td>Newbenefits</td>
<td>30</td>
<td>Stafford</td>
<td>30</td>
<td>4</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>DEPENDENT</th>
<th>ESSN</th>
<th>DEPENDENT_NAME</th>
<th>SEX</th>
<th>BDATE</th>
<th>RELATIONSHIP</th>
</tr>
</thead>
<tbody>
<tr>
<td>333445555</td>
<td>Alice</td>
<td>F</td>
<td>1986-04-05</td>
<td>DAUGHTER</td>
<td></td>
</tr>
<tr>
<td>333445555</td>
<td>Theodore</td>
<td>M</td>
<td>1983-10-26</td>
<td>SON</td>
<td></td>
</tr>
<tr>
<td>333445555</td>
<td>Joy</td>
<td>F</td>
<td>1958-05-03</td>
<td>SPOUSE</td>
<td></td>
</tr>
<tr>
<td>987654321</td>
<td>Abner</td>
<td>M</td>
<td>1942-02-28</td>
<td>SPOUSE</td>
<td></td>
</tr>
<tr>
<td>123456789</td>
<td>Michael</td>
<td>M</td>
<td>1988-01-04</td>
<td>SON</td>
<td></td>
</tr>
<tr>
<td>123456789</td>
<td>Alice</td>
<td>F</td>
<td>1988-12-30</td>
<td>DAUGHTER</td>
<td></td>
</tr>
<tr>
<td>123456789</td>
<td>Elizabeth</td>
<td>F</td>
<td>1967-05-05</td>
<td>SPOUSE</td>
<td></td>
</tr>
</tbody>
</table>
Query 0 (simple query)

- Retrieve the birthdate and address of the employee(s) whose last name is ‘Smith’

```
SELECT BDATE, ADDRESS
FROM EMPLOYEE
WHERE LNAME='Smith';
```

- Result

```
BDATE      ADDRESS
========== ==============================1965-01-09 731 Fondren, Houston, TX
```
Query 1 (select-project-join query)

- Retrieve the name and address of all employees who work for the ‘Research’ department

```sql
SELECT FNAME, LNAME, ADDRESS
FROM EMPLOYEE, DEPARTMENT
WHERE DNAME='Research' AND DNUMBER=DNO;
```

- Result

<table>
<thead>
<tr>
<th>FNAME</th>
<th>LNAME</th>
<th>ADDRESS</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Smith</td>
<td>731 Fondren, Houston, TX</td>
</tr>
<tr>
<td>Franklin</td>
<td>Wong</td>
<td>638 Voss, Houston, TX</td>
</tr>
<tr>
<td>Joyce</td>
<td>English</td>
<td>5631 Rice, Houston, TX</td>
</tr>
<tr>
<td>Ramesh</td>
<td>Narayan</td>
<td>975 Fire Oak, Humble, TX</td>
</tr>
</tbody>
</table>
Query 2 (more complex query)

- For every project located in ‘Stafford’, list the project number, the controlling department number, and the department manager’s last name, address, and birthdate

```
SELECT PNUMBER, DNUM, LNAME, ADDRESS, BDATE
FROM PROJECT, DEPARTMENT, EMPLOYEE
WHERE DNUM=DNUMBER AND MGRSSN=SSN AND PLOCATION='Stafford';
```

- Result

<table>
<thead>
<tr>
<th>PNUMBER</th>
<th>DNUM</th>
<th>LNAME</th>
<th>ADDRESS</th>
<th>BDATE</th>
</tr>
</thead>
<tbody>
<tr>
<td>10</td>
<td>4</td>
<td>Wallance</td>
<td>291 Berry, Bellaire, TX</td>
<td>1941-06-20</td>
</tr>
<tr>
<td>30</td>
<td>4</td>
<td>Wallance</td>
<td>291 Berry, Bellaire, TX</td>
<td>1941-06-20</td>
</tr>
</tbody>
</table>
SQL, Relational algebra, and Relational calculus

SELECT $A_1, A_2, \ldots, A_n$
FROM $R_1, R_2, \ldots, R_m$
WHERE $P$

- To Relational algebra:
  - $\pi_{A_1, A_2, \ldots, A_n}(\sigma_P(R_1 \times R_2 \times \ldots \times R_m))$

- To Relational calculus:
  - $\{t_1.A_1, t_2.A_2, \ldots, t_{m-k}.A_n \mid R_1(t_1) \land \ldots \land R_{m-k}(t_{m-k}) \land (\exists t_{m-k+1})(\exists t_m)(R_{m-k+1}(t_{m-k+1}) \land \ldots \land R_m(t_m) \land P)\}$
Query 0

SELECT BDATE, ADDRESS
FROM EMPLOYEE
WHERE FNAME='John' AND MINIT='B' AND LNAME='Smith';

- In Relational algebra
  - \( \pi_{BDATE, ADDRESS}(\sigma_{FNAME='John' \land MINIT='B' \land LNAME='Smith'}(EMPLOYEE)) \)

- In Relational calculus
  - \( \{t.BDATE, t.ADDRESS | EMPLOYEE(t) \land FNAME='John' \land MINIT='B' \land LNAME='Smith'\} \)
Query 1

```
SELECT          FNAME, LNAME, ADDRESS
FROM            EMPLOYEE, DEPARTMENT
WHERE           DNAME='Research' AND DNUMBER=DNO

- In Relational algebra
  - \( \pi_{\text{FNAME, LNAME, ADDRESS}} (\sigma_{\text{DNAME='Research' \land DNUMBER=DNO}} (\text{EMPLOYEE} \times \text{DEPARTMENT})) \)
  - \( \pi_{\text{FNAME, LNAME, ADDRESS}} (\sigma_{\text{DNAME='Research'}} (\text{EMPLOYEE} \bowtie_{\text{DNUMBER=DNO}} \text{DEPARTMENT})) \)

- In Relational calculus
  - \( \{ t.\text{FNAME}, t.\text{LNAME}, t.\text{ADDRESS} \mid \text{EMPLOYEE}(t) \land (\exists d)(\text{DEPARTMENT}(d) \land d.\text{DNAME}=\text{‘Research’} \land d.\text{DNUMBER}=t.\text{DNO}) \} \)
```
Queries without selection or projection

- Missing WHERE clause
  - No selection
  - All tuples of the table from FROM clause are selected
  - If more than 1 table, the result is Cross product

- Use of Asterisk (*)
  - No projection
  - Retrieves all attribute values of selected tuples
Query 3 (query without selection)

- Select all names of departments

```sql
SELECT DNAME
FROM DEPARTMENT;
```

- Result

```
DNAME
================
Headquarters
Administration
Research
```
Select all combinations of employees’ ssn and department names

**SELECT** SSN, DNAME **FROM** EMPLOYEE, DEPARTMENT;

<table>
<thead>
<tr>
<th>SSN</th>
<th>DNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>123456789</td>
<td>Headquarters</td>
</tr>
<tr>
<td>333445555</td>
<td>Headquarters</td>
</tr>
<tr>
<td>453453453</td>
<td>Headquarters</td>
</tr>
<tr>
<td>666884444</td>
<td>Headquarters</td>
</tr>
<tr>
<td>888665555</td>
<td>Headquarters</td>
</tr>
<tr>
<td>987654321</td>
<td>Headquarters</td>
</tr>
<tr>
<td>987987987</td>
<td>Headquarters</td>
</tr>
<tr>
<td>999887777</td>
<td>Headquarters</td>
</tr>
<tr>
<td>123456789</td>
<td>Administration</td>
</tr>
<tr>
<td>333445555</td>
<td>Administration</td>
</tr>
<tr>
<td>453453453</td>
<td>Administration</td>
</tr>
<tr>
<td>666884444</td>
<td>Administration</td>
</tr>
<tr>
<td>888665555</td>
<td>Administration</td>
</tr>
<tr>
<td>987654321</td>
<td>Administration</td>
</tr>
<tr>
<td>987987987</td>
<td>Administration</td>
</tr>
<tr>
<td>999887777</td>
<td>Administration</td>
</tr>
<tr>
<td>123456789</td>
<td>Research</td>
</tr>
<tr>
<td>333445555</td>
<td>Research</td>
</tr>
<tr>
<td>453453453</td>
<td>Research</td>
</tr>
<tr>
<td>666884444</td>
<td>Research</td>
</tr>
<tr>
<td>888665555</td>
<td>Research</td>
</tr>
<tr>
<td>987654321</td>
<td>Research</td>
</tr>
<tr>
<td>987987987</td>
<td>Research</td>
</tr>
<tr>
<td>999887777</td>
<td>Research</td>
</tr>
</tbody>
</table>
Query 5 (using asterisk)

- Retrieve all attribute values for employee named ‘Narayan’

\[
\begin{align*}
\text{SELECT } & * \\
\text{FROM } & \text{EMPLOYEE} \\
\text{WHERE } & \text{LNAME}= 'Narayan'
\end{align*}
\]

- Result

<table>
<thead>
<tr>
<th>FNAME</th>
<th>MINIT</th>
<th>LNAME</th>
<th>SSN</th>
<th>BDATE</th>
<th>ADDRESS</th>
<th>SEX</th>
<th>SALARY</th>
<th>SUPERSSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ramesh K</td>
<td>K</td>
<td>Narayan</td>
<td>666884444</td>
<td>1962-09-15</td>
<td>975 Fire Oak, Humble, TX</td>
<td>M</td>
<td>38000.00</td>
<td>333445555</td>
</tr>
</tbody>
</table>

5
Prefix attribute name with table name
- table_name.attribute_name
- in SELECT and WHERE clauses
  - same attribute names from different relations in a query

Introduce tuple variable for each relation
- table_name AS new_name
- in FROM clause
  - recursive query (join relation with itself)

Rename attribute name
- in SELECT clause
  - attribute_name AS new_name
Query 1A (Prefixing example)

- Suppose
  - LNAME of EMPLOYEE called NAME
  - DNAME of DEPARTMENT called NAME

- Retrieve the last name and address of all employees who work for the ‘Research’ department

```sql
SELECT EMPLOYEE.NAME, ADDRESS
FROM EMPLOYEE, DEPARTMENT
WHERE DEPARTMENT.NAME='Research' AND DNUMBER=DNO;
```
Query 6 (Tuple variables)

For each employee, retrieve the employee’s first and last name and the first and last name of his/her immediate supervisor.

```sql
SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME
FROM EMPLOYEE AS E, EMPLOYEE AS S
WHERE E.SUPERSSN=S.SSN;
```

<table>
<thead>
<tr>
<th>FNAME</th>
<th>LNAME</th>
<th>FNAME</th>
<th>LNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Smith</td>
<td>Franklin</td>
<td>Wong</td>
</tr>
<tr>
<td>Franklin</td>
<td>Wong</td>
<td>James</td>
<td>Borg</td>
</tr>
<tr>
<td>Joyce</td>
<td>English</td>
<td>Franklin</td>
<td>Wong</td>
</tr>
<tr>
<td>Ramesh</td>
<td>Narayan</td>
<td>Franklin</td>
<td>Wong</td>
</tr>
<tr>
<td>Jennifer</td>
<td>Wallace</td>
<td>James</td>
<td>Borg</td>
</tr>
<tr>
<td>Ahmad</td>
<td>Jabbar</td>
<td>Jennifer</td>
<td>Wallance</td>
</tr>
<tr>
<td>Alicia</td>
<td>Zelaya</td>
<td>Jennifer</td>
<td>Wallance</td>
</tr>
</tbody>
</table>
Query 6 (renaming, SELECT clause)

```sql
SELECT  E.FNAME AS E_FNAME, E.LNAME AS E_LNAME,
        S.FNAME AS S_FNAME, S.LNAME AS S_LNAME
FROM     EMPLOYEE AS E, EMPLOYEE AS S
WHERE    E.SUPERSSN=S.SSN;
```

Result

<table>
<thead>
<tr>
<th>E_FNAME</th>
<th>E_LNAME</th>
<th>S_FNAME</th>
<th>S_LNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

...
Duplicate elimination in SQL

- SQL does not automatically eliminates duplicates
  - it is expensive
  - user wants to use duplicates
  - when aggregate function is applied duplicates are wanted
    - could be specified explicitly by `SELECT ALL` …

- To eliminate duplicates specify
  - `SELECT DISTINCT` …
Query 7: retrieve the location of every project

```
SELECT PLOCATION
FROM PROJECT;
```

- **Result**
  - PLOCATION
  - Bellaire
  - Sugarland
  - Houston
  - Stafford
  - Houston
  - Stafford

```
SELECT DISTINCT PLOCATION
FROM PROJECT;
```

- **Result**
  - PLOCATION
  - Bellaire
  - Sugarland
  - Houston
  - Stafford
Set operation in SQL

- **Set operations**
  - UNION – set union
  - EXCEPT – set difference
  - INTERSECT – set intersection
    - table1 OP table2

- **Duplicates are eliminated**
  - use ALL to keep duplicates
    - UNION ALL, EXCEPT ALL, INTERSECT ALL

- **Applied only to union-compatible tables**
Query 8 (set operations)

Make a list of all project numbers for projects that involve an employee whose name is ‘Smith’, either as a worker or as a manager of the department that controls the project.

```sql
(SELECT DISTINCT PNUMBER
FROM PROJECT, DEPARTMENT, EMPLOYEE
WHERE DNUM=DNUMBER AND MGRSSN=SSN AND LNAME='Smith')
UNION
(SELECT DISTINCT PNUMBER
FROM PROJECT, WORKS_ON, EMPLOYEE
WHERE PNUMBER=PNO AND ESSN=SSN AND LNAME='Smith')
```

<table>
<thead>
<tr>
<th>PNUMBER</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
</tr>
<tr>
<td>2</td>
</tr>
</tbody>
</table>
Temporal data types

- **DATE** ‘yyyy-mm-dd’
- **TIME** ‘hh:mm:ss’
  - TIME WITH TIME ZONE ‘hh:mm:ss +hh:mm’
- **TIMESTAMP** ‘yyyy-mm-dd hh:mm:ss fffffff’
  - with time zone
  - e.g., TIMESTAMP ‘2002-09-27 09:12:47 648302’
- **INTERVAL** – a relative value
  - e.g., INTERVAL ‘1’ DAY
Operations

- **Arithmetic operators:**
  - addition (+), subtraction (-), multiplication (*), division (/)

- **String operator**
  - concatenation (||) of two strings

- **Temporal**
  - incrementing (+), decrementing (-) time, date, timestamp by interval data types

- **Can be used in SELECT and WHERE clauses**
  - use rename for result column with arithmetic operation
Query 9 (arithmetic operation)

- Show the resulting salaries if every employee working on the ‘ProductX’ project is given a 10 percent raise

```
SELECT FNAME, LNAME, 1.1*SALARY AS INC_SAL
FROM EMPLOYEE, WORKS_ON, PROJECT
WHERE SSN=ESSN AND PNO=PNUMBER AND PNAME='ProductX';
```

- Result

<table>
<thead>
<tr>
<th>FNAME</th>
<th>LNAME</th>
<th>INC_SAL</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Smith</td>
<td>33000.000</td>
</tr>
<tr>
<td>Joyce</td>
<td>English</td>
<td>27500.000</td>
</tr>
</tbody>
</table>
Specialized comparison operators

- Matching strings with patterns
  - use comparison operator \texttt{LIKE}
  - \% for any number of arbitrary symbols
  - \_ for any symbol

- Check that numerical value is inside an interval
  - Comparison operator \texttt{BETWEEN}
    - attribute \texttt{BETWEEN} value1 \texttt{AND} value2
    - (attribute $\geq$ value1) \texttt{AND} (attribute $\leq$ value2)
Query 10 (using LIKE)

- Retrieve all employees whose address is in Houston, Texas

```
SELECT FName, LNAME
FROM EMPLOYEE
WHERE ADDRESS LIKE '%Houston, TX%';
```

- Result

<table>
<thead>
<tr>
<th>FName</th>
<th>LName</th>
</tr>
</thead>
<tbody>
<tr>
<td>John</td>
<td>Smith</td>
</tr>
<tr>
<td>Franklin</td>
<td>Wong</td>
</tr>
<tr>
<td>James</td>
<td>Borg</td>
</tr>
<tr>
<td>Ahmad</td>
<td>Jabbar</td>
</tr>
</tbody>
</table>
Query 11 (using BETWEEN)

- Retrieve all employees in department 5 whose salary is between $30,000 and $40,000

```
SELECT LNAME, SALARY
FROM EMPLOYEE
WHERE (SALARY BETWEEN 30000 AND 40000)
AND DNO=5;
```

- Result

<table>
<thead>
<tr>
<th>LNAME</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith</td>
<td>30000.00</td>
</tr>
<tr>
<td>Wong</td>
<td>40000.00</td>
</tr>
<tr>
<td>Narayan</td>
<td>38000.00</td>
</tr>
</tbody>
</table>
Ordering result

- The tuples in the result can be ordered by the values of one or more attributes
  - use ORDER BY clause
  - tuples are ordered by first attribute than they are ordered within same value of the attribute by second attribute, and so on

- Order can be specified by
  - **ASC** – ascending order (default)
  - **DESC** – descending order
Query 12 (using ORDER BY)

- Retrieve a list of employees in the ascending order of their first name

```sql
SELECT FNAME, LNAME
FROM EMPLOYEE
ORDER BY FNAME;
```

### Result

<table>
<thead>
<tr>
<th>FNAME</th>
<th>LNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ahmad</td>
<td>Jabbar</td>
</tr>
<tr>
<td>Alicia</td>
<td>Zelaya</td>
</tr>
<tr>
<td>Franklin</td>
<td>Wong</td>
</tr>
<tr>
<td>James</td>
<td>Borg</td>
</tr>
<tr>
<td>Jennifer</td>
<td>Wallance</td>
</tr>
<tr>
<td>John</td>
<td>Smith</td>
</tr>
<tr>
<td>Joyce</td>
<td>English</td>
</tr>
<tr>
<td>Ramesh</td>
<td>Narayan</td>
</tr>
</tbody>
</table>
Query 13 (using DESC)

- Retrieve all employees and their salary ordered by their salary in descending order within each salary by their last name

```
SELECT FNAME, LNAME, SALARY
FROM EMPLOYEE
ORDER BY SALARY DESC, LNAME;
```

- Result

<table>
<thead>
<tr>
<th>FNAME</th>
<th>LNAME</th>
<th>SALARY</th>
</tr>
</thead>
<tbody>
<tr>
<td>James</td>
<td>Borg</td>
<td>55000.00</td>
</tr>
<tr>
<td>Jennifer</td>
<td>Wallance</td>
<td>43000.00</td>
</tr>
<tr>
<td>Franklin</td>
<td>Wong</td>
<td>40000.00</td>
</tr>
<tr>
<td>Ramesh</td>
<td>Narayan</td>
<td>38000.00</td>
</tr>
<tr>
<td>John</td>
<td>Smith</td>
<td>30000.00</td>
</tr>
<tr>
<td>Joyce</td>
<td>English</td>
<td>25000.00</td>
</tr>
<tr>
<td>Ahmad</td>
<td>Jabbar</td>
<td>25000.00</td>
</tr>
<tr>
<td>Alicia</td>
<td>Zelaya</td>
<td>25000.00</td>
</tr>
</tbody>
</table>
NULL Values

- Each NULL is unique (except grouping)
- Three-valued logic: TRUE, FALSE, UNKNOWN
- Result of queries contain only those row for which the condition is TRUE
- Check for NULL value
  - IS NULL
  - IS NOT NULL
Query 14 (using IS NULL)

- Retrieve the names of all employees who do not have supervisors

```
SELECT  FNAME, LNAME
FROM    EMPLOYEE
WHERE   SUPERSSN IS NULL;
```

- Result

<table>
<thead>
<tr>
<th>FNAME</th>
<th>LNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>James</td>
<td>Borg</td>
</tr>
</tbody>
</table>
### Three-valued logic: AND, OR, NOT

<table>
<thead>
<tr>
<th>AND</th>
<th>TRUE</th>
<th>FALSE</th>
<th>UNKNOWN</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>UNKNOWN</td>
</tr>
<tr>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
<td>FALSE</td>
</tr>
<tr>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
<td>FALSE</td>
<td>UNKNOWN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>OR</th>
<th>TRUE</th>
<th>FALSE</th>
<th>UNKNOWN</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
<td>TRUE</td>
</tr>
<tr>
<td>FALSE</td>
<td>TRUE</td>
<td>FALSE</td>
<td>UNKNOWN</td>
</tr>
<tr>
<td>UNKNOWN</td>
<td>TRUE</td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>NOT</th>
<th>TRUE</th>
<th>FALSE</th>
<th>UNKNOWN</th>
</tr>
</thead>
<tbody>
<tr>
<td>TRUE</td>
<td>FALSE</td>
<td>TRUE</td>
<td>UNKNOWN</td>
</tr>
<tr>
<td>FALSE</td>
<td>TRUE</td>
<td>UNKNOWN</td>
<td>UNKNOWN</td>
</tr>
</tbody>
</table>
Nested queries

- Complete select-from-where block (nested query) within **WHERE** clause of another query (outer query)
- Check if a tuple is contained by the result of nested query
  - attribute **IN** nested_query
  - **= ANY** and **= SOME**
- Comparison operators
  - >, <, >=, <=, =, <> with **ANY**, **SOME**, **ALL**
- Nested query is evaluated once for each tuple in the outer query
Query 15 (nested query)

- Retrieve SSN of all employees who work on the same project as employee with SSN=123456789

```sql
SELECT DISTINCT ESSN
FROM WORKS_ON
WHERE PNO IN (SELECT PNO
FROM WORKS_ON
WHERE ESSN='123456789');
```

- Result

<table>
<thead>
<tr>
<th>ESSN</th>
</tr>
</thead>
<tbody>
<tr>
<td>123456789</td>
</tr>
<tr>
<td>333445555</td>
</tr>
<tr>
<td>453453453</td>
</tr>
</tbody>
</table>

453453453
Query 16 (>ALL)

- Retrieve the names of employees whose salary is greater than the salary of all the employees in department 5

```
SELECT LNAME, FNAME
FROM EMPLOYEE
WHERE SALARY > ALL (SELECT SALARY
                      FROM EMPLOYEE
                      WHERE DNO=5);
```

- Result

```
<table>
<thead>
<tr>
<th>LNAME</th>
<th>FNAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Borg</td>
<td>James</td>
</tr>
<tr>
<td>Wallance</td>
<td>Jennifer</td>
</tr>
</tbody>
</table>
```
Nested queries

- Several levels of nested queries can be used
- Unqualified attribute refers to the relation declared in the innermost nested query
  - always create tuple variables to avoid potential errors and ambiguities
- Correlated nested queries
  - an attribute of outer query is referred in WHERE clause of nested query
- Queries written with nested query and using IN can be rewritten with single block query
Query 17 (correlated nested query)

- Retrieve the name of each employee who has a dependent with the same first name as the employee
  
  \[
  \text{SELECT } E.\text{FNAME}, E.\text{LNAME} \\
  \text{FROM } \text{EMPLOYEE AS E} \\
  \text{WHERE } E.\text{SSN} \text{ IN (SELECT ESSN} \\
  \text{FROM DEPENDENT} \\
  \text{WHERE E.FNAME=DEPENDENT_NAME)};
  \]

- Rewritten query
  
  \[
  \text{SELECT } E.\text{FNAME}, E.\text{LNAME} \\
  \text{FROM } \text{EMPLOYEE AS E, DEPENDENT AS D} \\
  \text{WHERE } E.\text{SSN}=D.\text{ESS AND} \\
  E.\text{FNAME}=D.\text{DEPENDENT_NAME};
  \]
EXISTS

- EXISTS checks if result of nested query is not empty
  - NOT EXISTS – opposite
- EXISTS are usually used in conjunction with correlated nested queries
SELECT  E.FNAME, E.LNAME
FROM    EMPLOYEE AS E
WHERE   EXISTS (SELECT *
               FROM  DEPENDEDENT
               WHERE E.SSN=ESSN AND
                      E.FNAME=DEPENDENT_NAME);
Query 18 (using NOT EXISTS)

- Retrieve the names of employee who have no dependents

```sql
SELECT  FNAME, LNAME
FROM    EMPLOYEE
WHERE   NOT EXISTS (SELECT *
                    FROM    DEPENDENT
                    WHERE   SSN=ESSN);
```
Join

- Query with more than one table
  - has always join between them
  - join conditions specified to avoid cross product
- Explicit join in FROM clause
  - to specify different type of join
  - to specify join condition together with join
  - `table1 (INNER) JOIN table2 ON join condition`
    - default
  - LEFT/RIGHT/FULL (OUTER) JOIN
  - NATURAL (INNER) JOIN
    - no condition (join on attributes with that same name)
    - with LEFT/RIGHT/FULL (OUTER) JOIN
Query 6 (using OUTER JOIN)

- Retrieve employee’s name with name of his supervisor

```
SELECT E.LNAME AS E_NAME, S.LNAME AS S_NAME
FROM (EMPLOYEE AS E
     LEFT OUTER JOIN
     EMPLOYEE AS S
     ON E.SUPERSSN = S.SSN);
```

<table>
<thead>
<tr>
<th>E_NAME</th>
<th>S_NAME</th>
</tr>
</thead>
<tbody>
<tr>
<td>Smith Wong</td>
<td>Wong</td>
</tr>
<tr>
<td>Wong Borg</td>
<td></td>
</tr>
<tr>
<td>English Wong</td>
<td></td>
</tr>
<tr>
<td>Narayan Wong</td>
<td></td>
</tr>
<tr>
<td>Borg -</td>
<td></td>
</tr>
<tr>
<td>Wallance Borg</td>
<td></td>
</tr>
<tr>
<td>Jabbar Wallance</td>
<td></td>
</tr>
<tr>
<td>Zelaya Wallance</td>
<td></td>
</tr>
</tbody>
</table>
Aggregate functions

- Functions
  - COUNT for rows
  - SUM, AVG numerical domain
  - MAX, MIN domains with total ordering
- NULL values discarded during applying aggregations on an attribute
- Used in SELECT and HAVING clauses
Query 19 (aggregate functions)

- Find the sum, max, min and avg of the salaries of all employees of the ‘Research’ department

```
SELECT SUM(SALARY), MAX(SALARY), MIN(SALARY), AVG(SALARY)
FROM EMPLOYEE, DEPARTMENT
WHERE DNAME='Research' AND DNO=DNUMBER;
```

- Result

<p>| | | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>133000.00</td>
<td>40000.00</td>
<td>25000.00</td>
<td>33250.00</td>
</tr>
</tbody>
</table>
Query 20 (using COUNT)

- Retrieve the total number of employees in the company

```
SELECT COUNT(*)
FROM EMPLOYEE;
```

- Result

```
8
```

Result

======
Grouping

- All result tuples are split to subgroups based on grouping attributes
  - Tuples are in the same subgroup if values of grouping attributes are the same
  - Separate subgroup for tuples with values NULL

- Grouping attributes defined in GROUP BY clause
  - Aggregate functions should be applied to all non-grouping attributes in SELECT clause
Query 21 (grouping example)

For each department, retrieve the department number, the number of employees in the department, and their average salary

```
SELECT DNO, COUNT(*), AVG(SALARY)
FROM EMPLOYEE
GROUP BY DNO;
```

Result

<table>
<thead>
<tr>
<th>DNO</th>
<th>Count</th>
<th>Average Salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>1</td>
<td>55000.000</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>31000.000</td>
</tr>
<tr>
<td>5</td>
<td>4</td>
<td>33250.000</td>
</tr>
</tbody>
</table>
Condition on group selection

- Retrieve groups that satisfy certain condition
  - in **HAVING** clause
- **HAVING** clause is used in conjunction with **GROUP BY** clause only
Query 22 (using HAVING)

- For each project on which more than two employees work, retrieve the project number, its name, and the number of its employees.

```sql
SELECT PNUMBER, PNAME, COUNT(*)
FROM PROJECT, WORKS_ON
WHERE PNUMBER=PNO
GROUP BY PNUMBER, PNAME
HAVING COUNT(*)>2;
```

- Result

<table>
<thead>
<tr>
<th>PNUMBER</th>
<th>PNAME</th>
<th>COUNT(*)</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>ProductY</td>
<td>3</td>
</tr>
<tr>
<td>10</td>
<td>Computerization</td>
<td>3</td>
</tr>
<tr>
<td>20</td>
<td>Reorganization</td>
<td>3</td>
</tr>
<tr>
<td>30</td>
<td>Newbenefits</td>
<td>3</td>
</tr>
</tbody>
</table>
Summary

- Clauses:
  
  ```sql
  SELECT <attribute list>
  FROM <table list>
  [WHERE <condition>]
  [GROUP BY <grouping attributes>]
  [HAVING <group condition>]
  [ORDER BY <attribute list>]
  ```

- Numerous ways to specify the same query