SQL for Querying and Updating

Acknowledgement:
Most of these slides have been adapted from slides made available by Pearson/Addison Wesley to accompany Elmasri and Navathe’s textbook Fundamentals of Database Systems

Outline

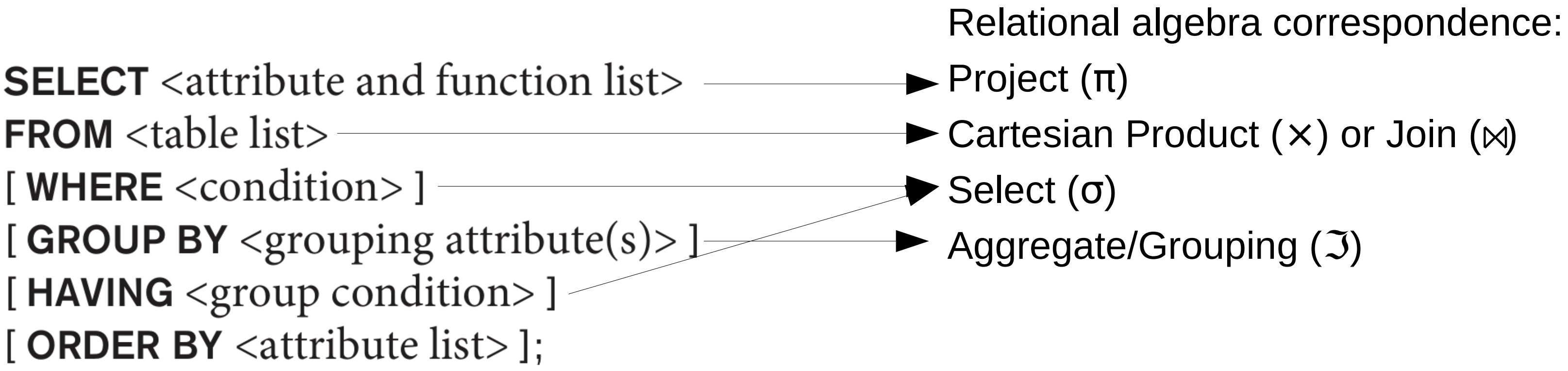
- The SELECT Structure of SQL Queries
  - Aliasing, and Renaming
  - Joined Tables
  - Aggregate Functions and Grouping
  - Tables as Sets in SQL
  - Nested Queries
  - Dealing with NULLs
  - The CASE Clause, and Result Limitation
- DELETE, and UPDATE Statements

Retrieval Queries in SQL

- SELECT statement
  - One basic statement for retrieving information from a database
- SQL allows a table to have two or more tuples that are identical in all their attribute values
  - Unlike relational model (relational model is strictly set-theory based)
  - Multiset or bag behavior
  - Tuple-id may be used as a key

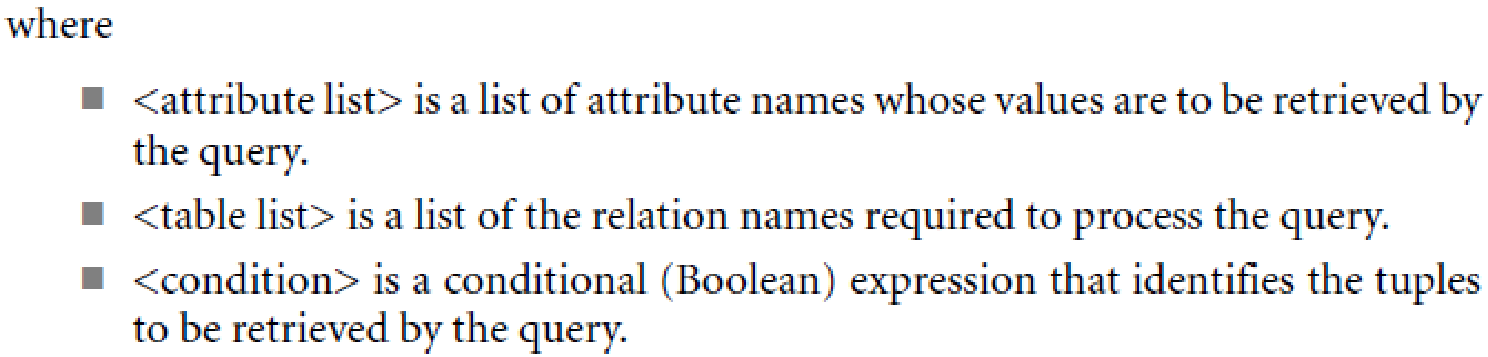
The SELECT Structure of SQL Queries

- Basic form of the SELECT statement:



Relational algebra correspondence:

SELECT <attribute and function list> → Project (π)
FROM <table list> → Cartesian Product (×) or Join (⋈)
[**WHERE** <condition>] → Select (σ)
[**GROUP BY** <grouping attribute(s)>] → Aggregate/Grouping (ℑ)
[**HAVING** <group condition>] → Select (σ)
[**ORDER BY** <attribute list>];



where

- <attribute list> is a list of attribute names whose values are to be retrieved by the query.
- <table list> is a list of the relation names required to process the query.
- <condition> is a conditional (Boolean) expression that identifies the tuples to be retrieved by the query.

The SELECT Structure of SQL Queries (2)

- Projection attributes
  - Attributes whose values are to be retrieved
  - Specify an asterisk (*) to retrieve all the attribute values
  - The projection can include functions and formulas
- Selection condition
  - Boolean condition that must be true for any retrieved tuple
  - Selection conditions include join conditions when multiple relations are involved

Logical and Arithmetic Operators, and Substring Pattern Matching

- Logical comparison operators: =, <, <=, >, >=, <>, etc.
- **BETWEEN** comparison operator
    - Example: **WHERE** Salary BETWEEN 30000 AND 40000
- **LIKE** comparison operator
  - Used for string **pattern matching**
  - % replaces an arbitrary number of zero or more characters
  - underscore (_) replaces a single character
    - Examples: **WHERE** Address **LIKE** '%Houston,TX%';
      WHERE Ssn **LIKE** '_ _ 1_ _ 8901';
- Standard arithmetic operators such as (+, –, *, /) and functions may be included as a part of SELECT
    - Example: **SELECT** salary * 0.9 **FROM** EMPLOYEE;
      WHERE age(Bdate) >= 21;

Ambiguous Attribute Names

- Same name can be used for two (or more) attributes in different relations
  - As long as the attributes are in different relations
  - Must **qualify** the attribute name with the relation name to prevent ambiguity
  - The * can be prefixed by the relation name to retrieve all attributes of a table

Example: Retrieve all attributes of the manager of the “Research” department

```
SELECT EMPLOYEE.*
FROM EMPLOYEE, DEPARTMENT
WHERE DEPARTMENT.Name=’Research’ AND
          DEPARTMENT.Dnumber=EMPLOYEE.Dnumber;
```

Aliasing, and Renaming

- Aliases or tuple variables
  - Are needed to refer to the same table more than once in a query
  - Can also be used just to shorten the query
- Renaming of attributes
  - Are needed to distinguish attributes that are homonyms
  - Can also be used for convenience

Example: For each employee, retrieve the employee’s first and last name and the first and last name of his or her immediate supervisor

```
SELECT  E.Fname, E.Lname,
                S.Fname AS Sup_Fname, S.Sname AS Sup_Sname
 FROM EMPLOYEE AS E, EMPLOYEE AS S
 WHERE E.Super_ssn=S.Ssn;
```

Aliasing, and Renaming (2)

- The attribute names can also be renamed in the tuple variable

Example:

```
SELECT  ...
FROM EMPLOYEE AS E(Fn, Mi, Ln, Ssn, Bd, Addr, Sex, Sal,
Sssn, Dno)...
```

- The “AS” may be dropped in most SQL implementations

Joined Tables

- The join operation can be represented by:
  - A cartesian product followed by a selection, or
  - A joined table
    - Specify different types of join
      - [INNER] JOIN, NATURAL JOIN, LEFT|RIGHT|FULL [OUTER] JOIN
    - Example: Retrieve all attributes of the manager of the “Research” department

```
SELECT e.*
FROM employee AS e JOIN department AS d
  ON e.dno=d.dnumber
WHERE d.name=’Research’;
```

Joined Tables (2)

- A joined table can nest join specifications
  - Optionally enclosed in parentheses
    - Example: Retrieve the first and last names of all employees and, for those who work on projects, the name of the project and the number of hours per week

```
SELECT e.fname, e.lname, p.pname, w.hours
FROM employee AS e
  LEFT OUTER JOIN works_on AS w ON e.ssn=w.essn
  LEFT OUTER JOIN project AS p ON w.pno=p.pnumber;
```

Aggregate Functions in SQL

- Used to summarize information from multiple tuples into a single-tuple summary
- Built-in aggregate functions
  - COUNT, SUM, MAX, MIN, AVG, STDDEV_POP, ...
  - NULLs are discarded
- Grouping
  - Create subgroups of tuples before summarizing
- To select entire groups, HAVING clause is used
  - HAVING corresponds to a posterior select operation
- Aggregate functions can be used in the SELECT clause or in a HAVING clause

Aggregate Functions in SQL (2)

- Examples
  - Retrieve the number of employees, the number of supervisees and the number supervisors

```
SELECT COUNT(*) AS num_employees,
  COUNT(super_ssn) AS num_supervisees,
  COUNT(DISTINCT(super_ssn)) AS num_supervisors
FROM employee;
```

  - Retrieve the average salary for employees per sex

```
SELECT sex, AVG(salary) AS avg_sal
FROM employee
GROUP BY sex;
```

  - Retrieve the names of the departments with at least 3 employees

```
SELECT d.dname
FROM department d JOIN employee e ON d.dnumber=e.dno
GROUP BY d.dname
HAVING COUNT(*) >= 3;
```

Aggregate Functions in SQL (3)

- Pitfalls when combining the WHERE and the HAVING Clause
  - Example: Retrieve the total number of employees whose salaries exceed $30,000 in each department, but only for departments where at least 3 employees work
  - **Incorrect** query:

```
SELECT dno, COUNT(*)
FROM employee
WHERE salary > 30000
GROUP BY dno
HAVING COUNT(*) >= 3;
```

  - **Correct** query:

```
SELECT dno, COUNT(*)
FROM employee
WHERE salary > 30000
  AND dno IN (SELECT dno FROM employee
                        GROUP BY dno HAVING HAVING COUNT(*) >= 3)
GROUP BY dno;
```

Ordering of Query Results

- Use **ORDER BY** clause
  - Keyword **DESC** to see result in a descending order of values
  - Keyword **ASC** to specify ascending order explicitly
  - Typically placed at the end of the query

```
SELECT ...
ORDER BY D.Dname DESC, E.Lname ASC,
                   E.Fname ASC;
```

Tables as Sets in SQL

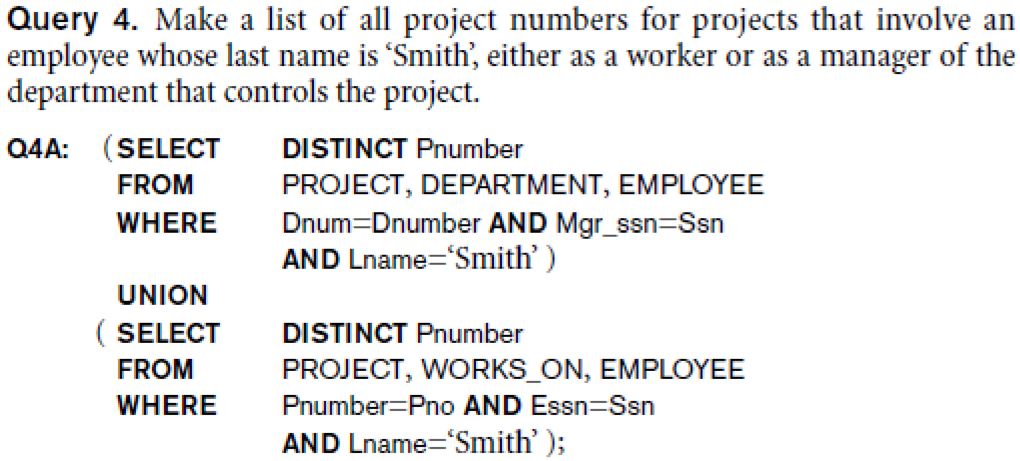
- SQL does not automatically eliminate duplicate tuples in query results
- Use the keyword **DISTINCT** in the SELECT clause
  - Only distinct tuples should remain in the result

Example:

```
SELECT DISTINCT E.SSN
FROM EMPLOYEE E, DEPENDENT D
WHERE E.SSN=D.ESSN;
```

Tables as Sets in SQL (2)

- Set operations
  - UNION, EXCEPT (difference), INTERSECT
    - Corresponding multiset operations: UNION ALL, EXCEPT ALL, INTERSECT ALL
  - Type compatibility is needed for these operations to be valid



Query 4. Make a list of all project numbers for projects that involve an employee whose last name is 'Smith', either as a worker or as a manager of the department that controls the project.

```
Q4A:  ( SELECT    DISTINCT Pnumber
        FROM      PROJECT, DEPARTMENT, EMPLOYEE
        WHERE     Dnum=Dnumber AND Mgr_ssn=Ssn
                  AND Lname='Smith' )
        UNION
      ( SELECT    DISTINCT Pnumber
        FROM      PROJECT, WORKS_ON, EMPLOYEE
        WHERE     Pnumber=Pno AND Essn=Ssn
                  AND Lname='Smith' );
```

Nested Queries

- Nested queries are complete “select-from-where-group by-having-order by” blocks within WHERE clause of another query (called the outer query)
- Comparison operators
  - IN (v IN S): evaluates to TRUE if v is one of the elements in the set S
    - S is usually the result of a nested query
  - ANY or SOME (v θ ANY S (or SOME)): returns TRUE if the value v is θ to some value in the set S
    - θ can be any of >, >=, <, <=, and <>
    - When θ is =, ANY/SOME behaves like the IN operator
  - ALL (v θ ALL S): returns TRUE if the value is θ of all values from S
  - EXISTS: evaluates to true if the result of the nested query is not empty
    - “For all” queries must use NOT EXISTS

Uncorrelated Nested Queries

- The nested query is evaluated only once
- Examples:
  - Retrieve the employees whose salary is greater than every salary from an employee who works for department 5

```
SELECT * FROM employee
WHERE salary > ALL (
  SELECT salary FROM employee
  WHERE dno=5);
```

  - Retrieve all projects that do not have any woman working on them

```
SELECT * FROM project
WHERE pnumber NOT IN (
  SELECT w.pno
  FROM employee e JOIN works_on w ON e.ssn=w.essn
  WHERE e.sex=’F’);
```

Correlated Nested Queries

- The nested query is evaluated once for each tuple in the outer query
- Examples:
  - Retrieve the employees who have no female dependent

```
SELECT * FROM employee e
WHERE NOT EXISTS (
  SELECT * FROM dependent d
  WHERE e.ssn=d.essn AND d.sex='F');
```

Representing the Relational Division using NOT EXISTS

- Example: Retrieve the names of employees who work on all the projects controlled by department 5

```
SELECT e.fname, e.minit, e.lname FROM employee e
WHERE NOT EXISTS (
  (SELECT p.pnumber FROM project p WHERE p.dnum=5)
  EXCEPT
  (SELECT w.pno FROM works_on w WHERE e.ssn= w.essn)
);

SELECT e.fname, e.minit, e.lname FROM employee e
WHERE NOT EXISTS (
  SELECT * FROM project p
  WHERE p.dnum=5 AND NOT EXISTS
        (SELECT * FROM works_on w WHERE w.essn=e.ssn AND w.pno=p.pnumber)
);
```

Comparisons Involving NULL and Three-Valued Logic

- Meanings of NULL
  - Unknown value
  - Unavailable or withheld value
  - Not applicable attribute
- Each individual NULL value considered to be different from every other NULL value
- SQL uses a three-valued logic:
  - TRUE, FALSE, and UNKNOWN (like Maybe)

Comparisons Involving NULL and Three-Valued Logic (2)

- Operations involving NULL return NULL
    - Examples:
        - NULL + 1 results in NULL
        - (balance + overdraft_limit) can be NULL
- Comparisons with NULL return UNKNOWN
    - Examples:
        - SELECT * FROM employee WHERE super_ssn=NULL;
            - Returns an empty result
        - SELECT * FROM employee WHERE sex<>'M';
            - Does not include in the result employees whose sex is either 'M' or NULL
- Check whether an attribute value is NULL (IS NULL or IS NOT NULL)
        - SELECT * FROM employee WHERE super_ssn IS NULL;
            - Returns the employees who do not have a direct supervisor

Comparisons Involving NULL and Three-Valued Logic (3)

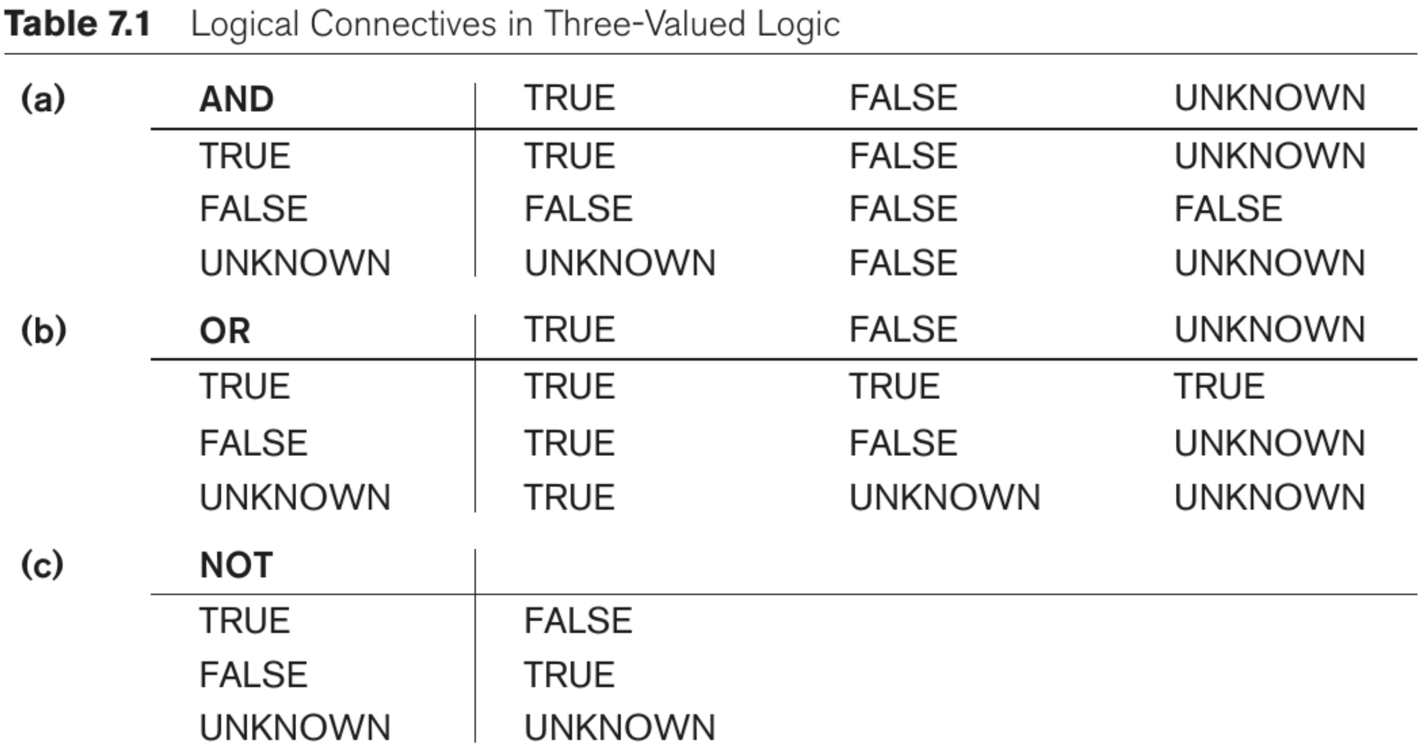


Table 7.1 Logical Connectives in Three-Valued Logic

(a)

AND	TRUE	FALSE	UNKNOWN
TRUE	TRUE	FALSE	UNKNOWN
FALSE	FALSE	FALSE	FALSE
UNKNOWN	UNKNOWN	FALSE	UNKNOWN

(b)

OR	TRUE	FALSE	UNKNOWN
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	UNKNOWN
UNKNOWN	TRUE	UNKNOWN	UNKNOWN

(c)

NOT	
TRUE	FALSE
FALSE	TRUE
UNKNOWN	UNKNOWN

Useful Clauses involving NULLs

- NULLIF
  - Returns NULL if the expressions are equal
  - Example:

    ```
    SELECT AVG(NULLIF(salary, 0.00)) AS avg_sal
    FROM employee;
    ```
- COALESCE
  - Returns the first non-NULL value in a list
  - Example:

    ```
    SELECT COALESCE(update_time, create_time,
    ‘Unknown’) AS last_updated
    FROM file_table;
    ```

The CASE Statement

- The CASE statement allows to define conditional instructions in SQL queries
- Syntax:

```
-- simple CASE
CASE ("column_name")
  WHEN "value1" THEN "result1"
  WHEN "value2" THEN "result2"
  ...
  [ELSE "resultN"]
END
```

```
-- searched CASE
SELECT CASE
  WHEN "condition1" THEN "result1"
  WHEN "condition2" THEN "result2"
  ...
  [ELSE "resultN"]
END
```

The CASE Statement (2)

- Example of CASE in queries:

```
SELECT ssn, fname, lname,
  CASE (sex)
    WHEN 'M' THEN 'Male'
    WHEN 'F' THEN 'Female'
    ELSE 'Other'
  END,
  CASE
    WHEN age(dt_nasc) >= 65 THEN 'Elder'
    WHEN age(dt_nasc) < 18 THEN 'Youth'
    ELSE 'Adult'
  END AS age_class
FROM employee;
```

The CASE Statement (3)

- Example of CASE in updates:

```
UPDATE employee
SET salary = (
  CASE
    WHEN dno=5 THEN salary*1.15
    WHEN dno=4 THEN salary*1.1
    ELSE salary*1.05
  END);
```

The CASE Statement (4)

- Example of CASE in aggregates:

```
SELECT dno,
  COUNT(
    CASE WHEN salary>30000 THEN 1
      ELSE NULL
    END) AS num_greater_30000,
  COUNT(
    CASE WHEN salary<=30000 THEN 1
      ELSE NULL
    END) AS num_upto_30000
FROM employee
GROUP BY dno;
```

Limiting the Size of the Result

- Useful for top-queries and pagination
  - SQL:2008: FETCH FIRST|NEXT k [PERCENT] ROWS + OFFSET m [PERCENT] ROWS
  - Proprietary extensions: TOP k, [OFFSET m] LIMIT k, etc.
- Example:

```
-- first page
SELECT ssn, fname, lname, salary
FROM employee
ORDER BY salary DESC
FETCH FIRST 10 ROWS ONLY; -- WITH TIES

-- second page
SELECT ssn, fname, lname, salary
FROM employee
ORDER BY salary DESC
OFFSET 10 ROWS
FETCH NEXT 10 ROWS ONLY; -- WITH TIES
```

DELETE, and UPDATE Statements in SQL

- In addition to INSERT, these are essential commands used to modify the database
  - These operations are executed in transactions, either explicitly stated by the user or implicitly done by the DBMS
    - BEGIN | START TRANSACTION;

      commands;

      COMMIT | ROLLBACK;
- UPDATE may update a number of tuples (rows) in a relation (table) that satisfy the condition
- DELETE may also update a number of tuples (rows) in a relation (table) that satisfy the condition

DELETE

- Removes tuples from a relation
    - Includes a WHERE-clause to select the tuples to be deleted
        - The number of tuples deleted depends on the number of tuples in the relation that satisfy the WHERE-clause
        - A missing WHERE-clause specifies that *all tuples* in the relation are to be deleted; the table then becomes an empty table
    - Referential integrity should be enforced
    - Tuples are deleted from only *one table* at a time
        - Unless CASCADE is specified on a referential integrity constraint

DELETE (2)

- Examples:
  - Delete the employees whose last name is Brown

```
DELETE FROM Employee WHERE Lname=’Brown’;
```

  - Delete all employees who work for the Administration department

```
DELETE FROM Employee WHERE Dno IN (
  SELECT Dnumber FROM Department
  WHERE Dname=’Administration’);
```

  - Delete all dependents

```
DELETE FROM Dependent;
```

UPDATE

- Used to modify attribute values of one or more selected tuples
  - A SET-clause specifies the attributes to be modified and their new values
  - A WHERE-clause selects the tuples to be modified
- Each command modifies tuples *in the same relation*
- All constraints should be enforced

UPDATE (2)

- Examples:
  - Change the location and controlling department number of project number 10 to 'Bellaire' and 5, respectively

```
UPDATE Project
  SET Plocation = 'Bellaire', DNUM = 5
  WHERE Pnumber = 10;
```

  - Give all employees in the department 5 a 10% raise in salary

```
UPDATE Employee
SET SALARY = SALARY*1.1
WHERE DNO = 5;
```

The RETURNING clause

- Update statements (INSERT, DELETE, UPDATE) can have a RETURNING clause, which returns the indicated attributes of the affected tuples
- Examples
  - Insert a project returning the identifier automatically genereated using a sequence

```
INSERT INTO project(pname, plocation, dnum)
  VALUES(‘Project XYZ’, ‘Stafford’, 5)
  RETURNING pnumber;
```

  - Give all employees in the department 5 a 10% raise in salary and return their names and new salaries

```
UPDATE Employee
  SET SALARY = SALARY*1.1
  WHERE DNO = 5 RETURNING fname, minit, lname, salary;
```