# Introducción a SQL

René F. Navarro

# Objetivos de SQL

- Idealmente, un lenguaje de base de datos debe permitir al usuario:
  - Crear la base de datos y las estructuras de las tablas;
  - Realizar operaciones básicas de gestión de los datos;
  - Realizar consultas simples y complejas.
- Debe realizar estas tareas requiriendo un esfuerzo mínimo del usuario y la sintaxis de los comandos debe ser fácil de aprender.
- Debe ser portable.

# Objetivos de SQL

- SQL es un lenguaje orientado a la transformación con 2 componentes principales:
  - Un DDL para definir la estructura de la base de datos.
  - Un DML para recuperar y actualizar datos.
- Hasta SQL: 1999, SQL no tenía comandos de flujo de control. Debían implementarse utilizando un lenguaje de programación, o interactivamente a decisión del usuario.

# Objetivos de SQL

- SQL es relativamente fácil de aprender:
  - No es procedural: se especifica qué información necesita, en lugar de cómo obtenerla;
  - Es esencialmente de formato libre.

# Objetivos de SQL

- Consiste en palabras de inglés estándar:
  - ```
    CREATE TABLE Staff (
          staffNo VARCHAR(5),
          lName VARCHAR(15),
          salary DECIMAL(7,2)
    );
    ```
  - ```
    INSERT INTO Staff VALUES (‘SG16’, ‘Brown’, 8300);
    ```
  - ```
    SELECT staffNo, lName, salary
          FROM Staff
          WHERE salary > 10000;
    ```
    ```

# Objetivos de SQL

- Puede ser utilizado por una variedad de usuarios, incluidos DBA, administración, desarrolladores de aplicaciones y otros tipos de usuarios finales.
- Ahora existe un estándar ISO para SQL, lo que lo convierte en el lenguaje estándar formal y de facto para bases de datos relacionales.

# Historia de SQL

- Todavía se pronuncia 'see-quel', aunque la pronunciación oficial es 'S-Q-L'.
- Posteriormente, IBM produjo un prototipo de DBMS llamado System R, basado en SEQUEL/2.
- Sin embargo, las raíces de SQL están en SQUARE (Specifying Queries as Relational Expressions), que es anterior al proyecto System R.

# Historia de SQL

| Año | Nombre | Comentarios |
|---|---|---|
| 1992 | SQL-92 | Revisión mayor. |
| 1999 | SQL:1999 | Se agregaron expresiones regulares, consultas recursivas (para relaciones jerárquicas), triggers y algunas características orientadas a objetos. |
| 2003 | SQL:2003 | Introduce algunas características de XML, cambios en las funciones, estandarización del objeto sequence y de las columnas autonuméricas.[4] |
| 2006 | SQL:2006 | ISO/IEC 9075-14:2006 Define las maneras en las cuales SQL se puede utilizar conjuntamente con XML. Define maneras de importar y guardar datos XML en una base de datos SQL, manipulándolos dentro de la base de datos y publicando el XML y los datos SQL convencionales en forma XML. Además, proporciona facilidades que permiten a las aplicaciones integrar dentro de su código SQL el uso de XQuery, lenguaje de consulta XML publicado por el W3C (World Wide Web Consortium) para acceso concurrente a datos ordinarios SQL y documentos XML. |
| 2008 | SQL:2008 | Permite el uso de la cláusula ORDER BY fuera de las definiciones de los cursores. Incluye los disparadores del tipo INSTEAD OF. Añade la sentencia TRUNCATE.[5] |
| 2011 | SQL:2011 | Datos temporales (PERIOD FOR). Mejoras en las funciones de ventana y de la cláusula FETCH. |
| 2016 | SQL:2016 | Permite búsqueda de patrones, funciones de tabla polimórficas y compatibilidad con los ficheros JSON. |

# Importancia de SQL

- SQL se ha convertido en parte de las arquitecturas de aplicaciones, como la arquitectura de aplicaciones de sistemas de IBM.
- Es la elección estratégica de muchas organizaciones grandes e influyentes (por ejemplo, X/OPEN).
- SQL es el Estándar Federal de Procesamiento de la Información (FIPS) al que se requiere conformidad para todas las ventas de bases de datos al Gobierno de los Estados Unidos.

# Escritura de comandos SQL

- Una instrucción SQL consta de palabras reservadas y palabras definidas por el usuario.
  - Las palabras reservadas son una parte fija de SQL y se deben escribir exactamente como se requiere y no se pueden dividir en líneas.
  - Las palabras definidas por el usuario se eligen por el usuario y representan nombres de varios objetos de la base de datos, como relaciones, columnas, vistas.

# Escritura de comandos SQL

- La mayoría de los componentes de una instrucción SQL no distinguen entre mayúsculas y minúsculas, excepto los datos de caracteres literales.
- Más legible con sangría y alineación:
  - Cada cláusula debe comenzar en una nueva línea.
  - El comienzo de una cláusula debe alinearse con el comienzo de otras cláusulas.
  - Si la cláusula tiene varias partes, cada una debe aparecer en una línea separada y estar sangrada al comienzo de la cláusula.

# Escritura de comandos SQL

- Use extended form of **Backus-Naur Form** (**BNF**) notation:
  - - Upper-case letters represent reserved words.
  - - Lower-case letters represent user-defined words.
  - - | indicates a choice among alternatives.
  - - {} Curly braces indicate a required element.
  - - [] Square brackets indicate an optional element.
  - - … indicates optional repetition (0 or more).

# Literales

- Los *literales* son constantes que se utilizan en las sentencias SQL.
- Todos los literales no numéricos deben ir entre comillas simples (por ejemplo, ‘Londres’).
- Todos los literales numéricos no deben estar entre comillas (por ejemplo, 650.00).

# SELECT Statement

```
SELECT [ DISTINCT | ALL ]
   { * | [ columnExpression [AS newName] ] [,...] }
FROM   TableName [alias] [, ...]
[ WHERE condition ]
[ GROUP BY  columnList ]
[ HAVING   condition]
[ ORDER BY  columnList ]
```

# SELECT Statement

| | |
|---|---|
| **SELECT** | Specifies which columns are to appear in output. |
| **FROM** | Specifies table(s) to be used. |
| **WHERE** | Filters rows. |
| **GROUP BY** | Forms groups of rows with same column value. |
| **HAVING** | Filters groups subject to some condition. |
| **ORDER BY** | Specifies the order of the output. |

# SELECT PostgreSQL

```
[ WITH [ RECURSIVE ] with_query [, ...] ]
SELECT [ ALL | DISTINCT [ ON ( expression [, ...] ) ] ]
    [ * | expression [ [ AS ] output_name ] [, ...] ]
    [ FROM from_item [, ...] ]
    [ WHERE condition ]
    [ GROUP BY [ ALL | DISTINCT ] grouping_element [, ...] ]
    [ HAVING condition ]
    [ WINDOW window_name AS ( window_definition ) [, ...] ]
    [ { UNION | INTERSECT | EXCEPT } [ ALL | DISTINCT ] select ]
    [ ORDER BY expression [ ASC | DESC | USING operator ] [ NULLS { FIRST | LAST } ] [, ...] ]
    [ LIMIT { count | ALL } ]
    [ OFFSET start [ ROW | ROWS ] ]
    [ FETCH { FIRST | NEXT } [ count ] { ROW | ROWS } { ONLY | WITH TIES } ]
    [ FOR { UPDATE | NO KEY UPDATE | SHARE | KEY SHARE } [ OF table_name [, ...] ] [ NOWAIT | SKIP LOCKED ] [...] ]
```

# SELECT PostgreSQL

```
where from_item can be one of:

    [ ONLY ] table_name [ * ] [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
                [ TABLESAMPLE sampling_method ( argument [, ...] ) [ REPEATABLE ( seed ) ] ]
    [ LATERAL ] ( select ) [ AS ] alias [ ( column_alias [, ...] ) ]
    with_query_name [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
    [ LATERAL ] function_name ( [ argument [, ...] ] )
                [ WITH ORDINALITY ] [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
    [ LATERAL ] function_name ( [ argument [, ...] ] ) [ AS ] alias ( column_definition [, ...] )
    [ LATERAL ] function_name ( [ argument [, ...] ] ) AS ( column_definition [, ...] )
    [ LATERAL ] ROWS FROM( function_name ( [ argument [, ...] ] ) [ AS ( column_definition [, ...] ) ] [, ...] )
                [ WITH ORDINALITY ] [ [ AS ] alias [ ( column_alias [, ...] ) ] ]
    from_item [ NATURAL ] join_type from_item [ ON join_condition | USING ( join_column [, ...] ) [ AS join_using_alias ] ]
```

# SELECT PostgreSQL

```
and grouping_element can be one of:

    ( )
    expression
    ( expression [, ...] )
    ROLLUP ( { expression | ( expression [, ...] ) } [, ...] )
    CUBE ( { expression | ( expression [, ...] ) } [, ...] )
    GROUPING SETS ( grouping_element [, ...] )

and with_query is:

    with_query_name [ ( column_name [, ...] ) ] AS [ [ NOT ] MATERIALIZED ] ( select | values | insert | update | delete )
        [ SEARCH { BREADTH | DEPTH } FIRST BY column_name [, ...] SET search_seq_col_name ]
        [ CYCLE column_name [, ...] SET cycle_mark_col_name [ TO cycle_mark_value DEFAULT cycle_mark_default ] USING cycle_path_col_name ]

TABLE [ ONLY ] table_name [ * ]
```

# SELECT Statement

- Order of the clauses cannot be changed.
- Only SELECT and FROM are mandatory.

# Example 5.1 All Columns, All Rows

List full details of all staff.

```
SELECT staffNo,fName,lName,address,position,sex,DOB,salary,branchNo
    FROM Staff;
```

Can use * as an abbreviation for ‘all columns’:

```
SELECT *
    FROM Staff;
```

# Example 5.1 All Columns, All Rows

**Table 5.1** Result table for Example 5.1.

| staffNo | fName | lName | position | sex | DOB | salary | branchNo |
|---|---|---|---|---|---|---|---|
| SL21 | John | White | Manager | M | 1-Oct-45 | 30000.00 | B005 |
| SG37 | Ann | Beech | Assistant | F | 10-Nov-60 | 12000.00 | B003 |
| SG14 | David | Ford | Supervisor | M | 24-Mar-58 | 18000.00 | B003 |
| SA9 | Mary | Howe | Assistant | F | 19-Feb-70 | 9000.00 | B007 |
| SG5 | Susan | Brand | Manager | F | 3-Jun-40 | 24000.00 | B003 |
| SL41 | Julie | Lee | Assistant | F | 13-Jun-65 | 9000.00 | B005 |

# Example 5.2 Specific Columns, All Rows

Produce a list of salaries for all staff, showing only staff number, first and last names, and salary.

```
SELECT staffNo, fName, lName, salary
    FROM Staff;
```

# Example 5.2 Specific Columns, All Rows

**Table 5.2** Result table for Example 5.2.

| staffNo | fName | lName | salary |
|---|---|---|---|
| SL21 | John | White | 30000.00 |
| SG37 | Ann | Beech | 12000.00 |
| SG14 | David | Ford | 18000.00 |
| SA9 | Mary | Howe | 9000.00 |
| SG5 | Susan | Brand | 24000.00 |
| SL41 | Julie | Lee | 9000.00 |

# Example 5.3 Use of DISTINCT

List the property numbers of all properties that have been viewed.

```
SELECT propertyNo
    FROM Viewing;
```

| propertyNo |
| --- |
| PA14 |
| PG4 |
| PG4 |
| PA14 |
| PG36 |

# Example 5.3 Use of DISTINCT

Use DISTINCT to eliminate duplicates:

```
SELECT DISTINCT propertyNo
    FROM Viewing;
```

| propertyNo |
| --- |
| PA14 |
| PG4 |
| PG36 |

# Example 5.4 Calculated Fields

Produce list of monthly salaries for all staff, showing staff number, first/last name, and salary.

```
SELECT staffNo, fName, lName, salary/12
        FROM Staff;
```

**Table 5.4** Result table for Example 5.4.

| staffNo | fName | lName | col4 |
|---|---|---|---|
| SL21 | John | White | 2500.00 |
| SG37 | Ann | Beech | 1000.00 |
| SG14 | David | Ford | 1500.00 |
| SA9 | Mary | Howe | 750.00 |
| SG5 | Susan | Brand | 2000.00 |
| SL41 | Julie | Lee | 750.00 |

# Example 5.4 Calculated Fields

To name column, use AS clause:

```
SELECT staffNo, fName, lName, salary/12 AS monthlySalary
    FROM Staff;
```

# Example 5.5 Comparison Search Condition

List all staff with a salary greater than 10,000.

```
SELECT staffNo, fName, lName, position, salary
    FROM Staff
    WHERE salary > 10000;
```

**Table 5.5** Result table for Example 5.5.

| staffNo | fName | lName | position | salary |
|---|---|---|---|---|
| SL21 | John | White | Manager | 30000.00 |
| SG37 | Ann | Beech | Assistant | 12000.00 |
| SG14 | David | Ford | Supervisor | 18000.00 |
| SG5 | Susan | Brand | Manager | 24000.00 |

# Example 5.6 Compound Comparison Search Condition

List addresses of all branch offices in London or Glasgow.

```
SELECT *
    FROM Branch
    WHERE city = 'London' OR city = 'Glasgow';
```

**Table 5.6** Result table for Example 5.6.

| branchNo | street | city | postcode |
|---|---|---|---|
| B005 | 22 Deer Rd | London | SW1 4EH |
| B003 | 163 Main St | Glasgow | G11 9QX |
| B002 | 56 Clover Dr | London | NW10 6EU |

# Example 5.7 Range Search Condition

List all staff with a salary between 20,000 and 30,000.

```
SELECT staffNo, fName, lName, position, salary
    FROM Staff
    WHERE salary BETWEEN 20000 AND 30000;
```

BETWEEN test includes the endpoints of range.

# Example 5.7 Range Search Condition

**Table 5.7** Result table for Example 5.7.

| staffNo | fName | lName | position | salary |
|---|---|---|---|---|
| SL21 | John | White | Manager | 30000.00 |
| SG5 | Susan | Brand | Manager | 24000.00 |

# Example 5.7 Range Search Condition

Also a negated version NOT BETWEEN.

BETWEEN does not add much to SQL's expressive power. Could also write:

```
SELECT staffNo, fName, lName, position, salary
    FROM Staff
    WHERE salary>=20000 AND salary <= 30000;
```

Useful, though, for a range of values.

# Example 5.8 Set Membership

List all managers and supervisors.

```
SELECT staffNo, fName, lName, position
    FROM Staff
    WHERE position IN ('Manager', 'Supervisor');
```

**Table 5.8** Result table for Example 5.8.

| staffNo | fName | lName | position |
|---|---|---|---|
| SL21 | John | White | Manager |
| SG14 | David | Ford | Supervisor |
| SG5 | Susan | Brand | Manager |

# Example 5.8 Set Membership

- There is a negated version (NOT IN).
- IN does not add much to SQL's expressive power. Could have expressed this as:

```
SELECT staffNo, fName, lName, position
  FROM Staff
  WHERE position='Manager' OR
                  position='Supervisor';
```

- IN is more efficient when set contains many values.

# Example 5.9 Pattern Matching

Find all owners with the string 'Glasgow' in their address.

```
SELECT ownerNo, fName, lName, address, telNo
  FROM PrivateOwner
  WHERE address LIKE '%Glasgow%';
```

**Table 5.9** Result table for Example 5.9.

| ownerNo | fName | lName | address | telNo |
|---|---|---|---|---|
| CO87 | Carol | Farrel | 6 Achray St, Glasgow G32 9DX | 0141-357-7419 |
| CO40 | Tina | Murphy | 63 Well St, Glasgow G42 | 0141-943-1728 |
| CO93 | Tony | Shaw | 12 Park Pl, Glasgow G4 0QR | 0141-225-7025 |

# Example 5.9 Pattern Matching

- SQL has two special pattern matching symbols:
  - **%** sequence of zero or more characters;
  - _ any single character.
- LIKE ‘%Glasgow%’ means a sequence of characters of any length containing ‘*Glasgow*’.

# Example 5.10 NULL Search Condition

List details of all viewings on property PG4 where a comment has not been supplied.

- There are 2 viewings for property PG4, one with and one without a comment.
- Have to test for null explicitly using special keyword IS NULL:

```
SELECT clientNo, viewDate
    FROM Viewing
    WHERE propertyNo = 'PG4' AND  comment IS NULL;
```

# Example 5.11 Single Column Ordering

List salaries for all staff, arranged in descending order of salary.

```
SELECT staffNo, fName, lName, salary
    FROM Staff
    ORDER BY salary DESC;
```

# Example 5.11 Single Column Ordering

**Table 5.11** Result table for Example 5.11.

| staffNo | fName | lName | salary |
|---|---|---|---|
| SL21 | John | White | 30000.00 |
| SG5 | Susan | Brand | 24000.00 |
| SG14 | David | Ford | 18000.00 |
| SG37 | Ann | Beech | 12000.00 |
| SA9 | Mary | Howe | 9000.00 |
| SL41 | Julie | Lee | 9000.00 |

# Example 5.12 Multiple Column Ordering

Produce abbreviated list of properties in order of property type.

```
SELECT propertyNo, type, rooms, rent
    FROM PropertyForRent
    ORDER BY type;
```

# Example 5.12 Multiple Column Ordering

**Table 5.12(a)** Result table for Example 5.12 with one sort key.

| propertyNo | type | rooms | rent |
|---|---|---|---|
| PL94 | Flat | 4 | 400 |
| PG4 | Flat | 3 | 350 |
| PG36 | Flat | 3 | 375 |
| PG16 | Flat | 4 | 450 |
| PA14 | House | 6 | 650 |
| PG21 | House | 5 | 600 |

# Example 5.12 Multiple Column Ordering

- Four flats in this list - as no minor sort key specified, system arranges these rows in any order it chooses.
- To arrange in order of rent, specify minor order:

```
SELECT propertyNo, type, rooms, rent
    FROM PropertyForRent
    ORDER BY type, rent DESC;
```

# Example 5.12 Multiple Column Ordering

**Table 5.12(b)** Result table for Example 5.12 with two sort keys.

| propertyNo | type | rooms | rent |
|---|---|---|---|
| PG16 | Flat | 4 | 450 |
| PL94 | Flat | 4 | 400 |
| PG36 | Flat | 3 | 375 |
| PG4 | Flat | 3 | 350 |
| PA14 | House | 6 | 650 |
| PG21 | House | 5 | 600 |

# SELECT Statement - Aggregates

ISO standard defines five aggregate functions:

- COUNT returns number of values in specified column.
- SUM returns sum of values in specified column.
- AVG returns average of values in specified column.
- MIN returns smallest value in specified column.
- MAX returns largest value in specified column.

# SELECT Statement - Aggregates

- Cada uno opera en una sola columna de una tabla y devuelve un solo valor.
- COUNT, MIN y MAX se aplican a campos numéricos y no numéricos, pero SUM y AVG solo se pueden usar en campos numéricos.
- Aparte de COUNT(*), cada función elimina los valores nulos primero y opera solo con los valores no nulos restantes.

# SELECT Statement - Aggregates

- COUNT(*) cuenta todas las filas de una tabla, independientemente de si se producen valores nulos o duplicados.
- Puede usar DISTINCT antes del nombre de la columna para eliminar duplicados.
- DISTINCT no tiene ningún efecto con MIN/MAX, pero puede tenerlo con SUM/AVG.

# SELECT Statement - Aggregates

- Las funciones agregadas solo se pueden usar en la lista SELECT y en la cláusula HAVING.
- Si SELECT incluye una función agregada y no hay una cláusula GROUP BY, la lista SELECT no puede hacer referencia a una columna con una función agregada. Por ejemplo, lo siguiente es ilegal:

```
SELECT staffNo, COUNT(salary)
   FROM Staff;
```

# Example 5.13 Use of COUNT(*)

How many properties cost more than £350 per month to rent?

```
SELECT COUNT(*) AS myCount
       FROM PropertyForRent
       WHERE rent > 350;
```

# Example 5.14 Use of COUNT(DISTINCT)

How many different properties viewed in May ‘04?

```
SELECT COUNT(DISTINCT propertyNo) AS myCount
 FROM Viewing
 WHERE viewDate BETWEEN ‘1-May-04’    AND ‘31-May-04’;
```

| myCount |
| --- |
| 2 |

# Example 5.15 Use of COUNT and SUM

Find number of Managers and sum of their salaries.

```
SELECT COUNT(staffNo) AS myCount, SUM(salary) AS mySum
    FROM Staff
    WHERE position = ‘Manager’;
```

# Example 5.16 Use of MIN, MAX, AVG

Find minimum, maximum, and average staff salary.

```
SELECT MIN(salary) AS myMin, MAX(salary) AS myMax, AVG(salary) AS myAvg
       FROM Staff;
```

# SELECT Statement - Grouping

- Cláusula **GROUP BY** para obtener subtotales.
- **SELECT** y **GROUP BY** están estrechamente integrados: cada elemento de la lista SELECT debe tener un solo valor por grupo, y la cláusula SELECT sólo puede contener:
  - nombres de columnas
  - Funciones agregadas
  - constantes
  - expresión que involucra combinaciones de las anteriores.

# SELECT Statement - Grouping

- Todos los nombres de columnas en la lista SELECT deben aparecer en la cláusula GROUP BY, a menos que el nombre se use solo en una función agregada.
- Si se usa WHERE con GROUP BY, WHERE se aplica primero, luego los grupos se forman a partir de las filas restantes que satisfacen el predicado.
- ISO considera que dos valores nulos son iguales a efectos de GROUP BY.

# Example 5.17 Use of GROUP BY

Find number of staff in each branch and their total salaries.

```
SELECT branchNo,COUNT(staffNo) AS myCount,SUM(salary) AS
mySum
      FROM Staff
      GROUP BY branchNo
      ORDER BY branchNo;
```

# Example 5.17 Use of GROUP BY

| branchNo | myCount | mySum |
|---|---|---|
| B003 | 3 | 54000.00 |
| B005 | 2 | 39000.00 |
| B007 | 1 | 9000.00 |

# Restricted Groupings – HAVING clause

- La cláusula HAVING está diseñada para usarse con GROUP BY para restringir los grupos que aparecen en la tabla de resultados final.
- Similar a WHERE, pero WHERE filtra filas individuales mientras que HAVING filtra grupos.
- Los nombres de las columnas en la cláusula HAVING también deben aparecer en la lista GROUP BY o estar contenidos dentro de una función agregada.

# Example 5.18 Use of HAVING

For each branch with more than 1 member of staff, find number of staff in each branch and sum of their salaries.

```
SELECT branchNo,  COUNT(staffNo) AS myCount,
                SUM(salary) AS mySum
     FROM Staff
     GROUP BY branchNo
     HAVING COUNT(staffNo) > 1
     ORDER BY branchNo;
```

# Example 5.18 Use of HAVING

| branchNo | myCount | mySum |
|---|---|---|
| B003 | 3 | 54000.00 |
| B005 | 2 | 39000.00 |

# Subqueries

- Algunas sentencias SQL pueden tener un SELECT incrustado en ellas.
- Se puede utilizar una *subselección* en las cláusulas WHERE y HAVING de un SELECT externo, donde se denomina subconsulta o consulta anidada.
- Las *subselecciones* también pueden aparecer en declaraciones INSERT, UPDATE y DELETE.

# Example 5.19 Subquery with Equality

List staff who work in branch at ‘163 Main St’.

```
SELECT staffNo, fName, lName, position
    FROM Staff
    WHERE branchNo =
        (SELECT branchNo
         FROM Branch
         WHERE street = ‘163 Main St’);
```

# Example 5.19 Subquery with Equality

- Inner SELECT finds branch number for branch at ‘163 Main St’ (‘B003’).
- Outer SELECT then retrieves details of all staff who work at this branch.
- Outer SELECT then becomes:

```
SELECT staffNo, fName, lName, position
   FROM Staff
   WHERE branchNo = ‘B003’;
```

# Example 5.19 Subquery with Equality

**Table 5.19** Result table for Example 5.19.

| staffNo | fName | lName | position |
|---|---|---|---|
| SG37 | Ann | Beech | Assistant |
| SG14 | David | Ford | Supervisor |
| SG5 | Susan | Brand | Manager |

# Example 5.20 Subquery with Aggregate

List all staff whose salary is greater than the average salary, and show by how much.

```
SELECT staffNo, fName, lName, position,
     salary – (SELECT AVG(salary) FROM Staff) As sd
      FROM Staff
      WHERE salary > (SELECT AVG(salary) FROM
        Staff) ;
```

# Example 5.20 Subquery with Aggregate

- Cannot write ‘WHERE salary > AVG(salary)’
- Instead, use subquery to find average salary (17000), and then use outer SELECT to find those staff with salary greater than this:

```
SELECT staffNo, fName, lName, position,   salary – 17000 As sd
      FROM Staff
      WHERE salary > 17000;
```

# Example 5.20 Subquery with Aggregate

**Table 5.20** Result table for Example 5.20.

| staffNo | fName | lName | position | salDiff |
|---|---|---|---|---|
| SL21 | John | White | Manager | 13000.00 |
| SG14 | David | Ford | Supervisor | 1000.00 |
| SG5 | Susan | Brand | Manager | 7000.00 |

# Subquery Rules

- ORDER BY clause may not be used in a subquery (although it may be used in outermost SELECT).
- Subquery SELECT list must consist of a single column name or expression, except for subqueries that use EXISTS.
- By default, column names refer to table name in FROM clause of subquery. Can refer to a table in FROM using an *alias*.
- When subquery is an operand in a comparison, subquery must appear on right-hand side.
- A subquery may not be used as an operand in an expression.

# Example 5.21 Nested subquery: use of IN

List properties handled by staff at ‘163 Main St’.

```
SELECT propertyNo, street, city, postcode, type, rooms, rent
      FROM PropertyForRent
      WHERE staffNo IN (SELECT staffNo
                        FROM Staff
                        WHERE branchNo =  (SELECT branchNo
                                           FROM Branch
                                           WHERE street = ‘163 Main St’) );
```

# Example 5.21 Nested subquery: use of IN

**Table 5.21** Result table for Example 5.21.

| propertyNo | street | city | postcode | type | rooms | rent |
|---|---|---|---|---|---|---|
| PG16 | 5 Novar Dr | Glasgow | G12 9AX | Flat | 4 | 450 |
| PG36 | 2 Manor Rd | Glasgow | G32 4QX | Flat | 3 | 375 |
| PG21 | 18 Dale Rd | Glasgow | G12 | House | 5 | 600 |

# ANY and ALL

- ANY and ALL may be used with subqueries that produce a single column of numbers.
- With ALL, condition will only be true if it is satisfied by *all* values produced by subquery.
- With ANY, condition will be true if it is satisfied by *any* values produced by subquery.
- If subquery is empty, ALL returns true, ANY returns false.
- SOME may be used in place of ANY.

# Example 5.22 Use of ANY/SOME

Find staff whose salary is larger than salary of at least one member of staff at branch B003.

```
SELECT staffNo, fName, lName, position, salary
      FROM Staff
      WHERE salary > SOME ( SELECT salary
                            FROM Staff
                            WHERE branchNo = ‘B003’ );
```

# Example 5.22 Use of ANY/SOME

- Inner query produces set {12000, 18000, 24000} and outer query selects those staff whose salaries are greater than any of the values in this set.

# Example 5.23 Use of ALL

Find staff whose salary is larger than salary of every member of staff at branch B003.

```
SELECT staffNo, fName, lName, position, salary
 FROM Staff
 WHERE salary > ALL  ( SELECT salary
                       FROM Staff
                       WHERE branchNo = ‘B003’);
```

# Example 5.23 Use of ALL

**Table 5.23** Result table for Example 5.23.

| staffNo | fName | lName | position | salary |
|---|---|---|---|---|
| SL21 | John | White | Manager | 30000.00 |

# Consultas multitablas

- Se pueden usar subconsultas si los resultados proveen columnas resultados de la misma tabla.
- Si los resultados vienen de más de una tabla, se debe usar una combinación (join).
- Para realizar un join, se incluye más de una tabla en la cláusula FROM.
- También es posible usar un alias de una tabla nombrada en la cláusula FROM.
- El alias se separa del nombre de la tabla con un espacio.
- El alias se puede usar para calificar los nombres de columnas cuando exista ambigüedad.

# Example 5.24 Simple Join

Nombres de todos los clientes que han visto alguna propiedad junto con sus comentarios.

```
SELECT c.clientNo, fName, lName, propertyNo, comment
      FROM Client c, Viewing v
      WHERE c.clientNo = v.clientNo;
```

# Example 5.24 Simple Join

- Solo se incluyen los renglones de ambas tablas que tengan valores idénticos en las columnas clientNo (c.clientNo = v.clientNo) en los resultados.
- Equivalente al equijoin.

**Table 5.24** Result table for Example 5.24.

| clientNo | fName | lName | propertyNo | comment |
|---|---|---|---|---|
| CR56 | Aline | Stewart | PG36 | |
| CR56 | Aline | Stewart | PA14 | too small |
| CR56 | Aline | Stewart | PG4 | |
| CR62 | Mary | Tregear | PA14 | no dining room |
| CR76 | John | Kay | PG4 | too remote |

# Formas alternas del JOIN

- Formas alternativas del join:

```
FROM Client c JOIN Viewing v ON c.clientNo = v.clientNo
FROM Client JOIN Viewing USING (clientNo)
FROM Client NATURAL JOIN Viewing
```

- En cada caso, FROM reemplaza a las cláusulas originales FROM y WHERE. Sin embargo, el primero produce una tabla con dos columnas idénticas clientNo.

# Example 5.25 Sorting a join

Para cada sucursal, listar números y nombres del staff que administra propiedades y las propiedades que administra.

```
SELECT s.branchNo, s.staffNo, fName, lName,
                  propertyNo
   FROM Staff s, PropertyForRent p
   WHERE s.staffNo = p.staffNo
   ORDER BY s.branchNo, s.staffNo, propertyNo;
```

# Example 5.25 Sorting a join

**Table 5.25** Result table for Example 5.25.

| branchNo | staffNo | fName | lName | propertyNo |
|---|---|---|---|---|
| B003 | SG14 | David | Ford | PG16 |
| B003 | SG37 | Ann | Beech | PG21 |
| B003 | SG37 | Ann | Beech | PG36 |
| B005 | SL41 | Julie | Lee | PL94 |
| B007 | SA9 | Mary | Howe | PA14 |

# Example 5.26 Three Table Join

For each branch, list staff who manage properties, including city in which branch is located and properties they manage.

```
SELECT b.branchNo, b.city, s.staffNo, fName, lName,
                        propertyNo
        FROM Branch b, Staff s, PropertyForRent p
        WHERE b.branchNo = s.branchNo AND
                        s.staffNo = p.staffNo
        ORDER BY b.branchNo, s.staffNo, propertyNo;
```

# Example 5.26 Three Table Join

**Table 5.26** Result table for Example 5.26.

| branchNo | city | staffNo | fName | lName | propertyNo |
|---|---|---|---|---|---|
| B003 | Glasgow | SG14 | David | Ford | PG16 |
| B003 | Glasgow | SG37 | Ann | Beech | PG21 |
| B003 | Glasgow | SG37 | Ann | Beech | PG36 |
| B005 | London | SL41 | Julie | Lee | PL94 |
| B007 | Aberdeen | SA9 | Mary | Howe | PA14 |

- Alternative formulation for FROM and WHERE:

```
FROM (Branch b JOIN Staff s USING branchNo) AS
          bs JOIN PropertyForRent p USING staffNo
```

# Example 5.27 Multiple Grouping Columns

Find number of properties handled by each staff member.

```
SELECT s.branchNo, s.staffNo, COUNT(*) AS myCount
   FROM Staff s, PropertyForRent p
   WHERE s.staffNo = p.staffNo
   GROUP BY s.branchNo, s.staffNo
   ORDER BY s.branchNo, s.staffNo;
```

# Example 5.27 Multiple Grouping Columns

| branchNo | staffNo | myCount |
|---|---|---|
| B003 | SG14 | 1 |
| B003 | SG37 | 2 |
| B005 | SL41 | 1 |
| B007 | SA9 | 1 |

# Calculando un JOIN

Para generar los resultados del *join*:

1. Formar el producto cartesiano de las tablas nombradas en la cláusula FROM.
2. Si hay una cláusula WHERE, aplicar la condición de búsqueda a cada renglón de la tabla producto, reteniendo aquellos renglones que satisfacen la condición.
3. Para renglón restante, determinar el valor de cada elemento de la lista SELECT para generar un renglón de la tabla de resultados.
4. Si se ha especificado SELECT DISTINCT, eliminar renglones duplicados.
5. Si hay una cláusula ORDER BY, ordenar los resultados según se requiera.

# Outer Joins

- Si un renglón de la tabla combinada no cuadra, el renglón se omite de la tabla resultado.
- Las operaciones de **combinación externa** (*outer join*) retienen renglones que no satisfacen la condición del *join*.
- Considerar las siguientes tablas:

Branch1

| branchNo | bCity |
|---|---|
| B003 | Glasgow |
| B004 | Bristol |
| B002 | London |

PropertyForRent1

| propertyNo | pCity |
|---|---|
| PA14 | Aberdeen |
| PL94 | London |
| PG4 | Glasgow |

# Outer Joins

- El join (interno) de las dos tablas es:

```
SELECT b.*, p.*
 FROM Branch1 b, PropertyForRent1 p
 WHERE b.bCity = p.pCity;
```

**Table 5.27(b)** Result table for inner join of Branch1 and PropertyForRent1 tables.

| branchNo | bCity | propertyNo | pCity |
|---|---|---|---|
| B003 | Glasgow | PG4 | Glasgow |
| B002 | London | PL94 | London |

# Outer Joins

- La tabla de resultados tiene dos renglones donde las ciudades son las mismas.
- No hay renglones que correspondan a las sucursales de Bristol y Aberdeen.
- Para incluir renglones que no cuadran, usar un Outer join.

# Example 5.28 Left Outer Join

Listar sucursales y propiedades que están en la misma ciudad junto con cualquier sucursal que no cuadre.

```
SELECT b.*, p.*
   FROM Branch1 b LEFT JOIN
        ProertyForRent1 p ON b.bCity = p.pCity;
```

# Example 5.28 Left Outer Join

- Incluye aquellos renglones de la primera tabla (left) que no cuadran con renglones de la segunda tabla (right).
- Columnas de la segunda tabla se rellenan con NULL.

**Table 5.28** Result table for Example 5.28.

| branchNo | bCity | propertyNo | pCity |
|---|---|---|---|
| B003 | Glasgow | PG4 | Glasgow |
| B004 | Bristol | NULL | NULL |
| B002 | London | PL94 | London |

# Example 5.29 Right Outer Join

Listar sucursales y propiedades en la misma ciudad y cualquier propiedad que no cuadre.

```
SELECT b.*, p.*
        FROM Branch1 b RIGHT JOIN
        PropertyForRent1 p ON b.bCity = p.pCity;
```

# Example 5.29 Right Outer Join

- Incluye aquellos renglones de la segunda tabla (right) que no cuadran con renglones de la primera tabla (left).
- Columnas de la primera tabla se rellenan con NULL.

**Table 5.29** Result table for Example 5.29.

| branchNo | bCity | propertyNo | pCity |
|---|---|---|---|
| NULL | NULL | PA14 | Aberdeen |
| B003 | Glasgow | PG4 | Glasgow |
| B002 | London | PL94 | London |

# Example 5.30 Full Outer Join

List branches and properties in same city and any unmatched branches or properties.

```
SELECT b.*, p.*
        FROM Branch1 b FULL JOIN
        PropertyForRent1 p ON b.bCity = p.pCity;
```

# Example 5.30 Full Outer Join

- Renglones que no cuadran de las tablas.
- A columnas que no cuadren asigna NULL.

**Table 5.30** Result table for Example 5.30.

| branchNo | bCity | propertyNo | pCity |
|---|---|---|---|
| NULL | NULL | PA14 | Aberdeen |
| B003 | Glasgow | PG4 | Glasgow |
| B004 | Bristol | NULL | NULL |
| B002 | London | PL94 | London |

# EXISTS and NOT EXISTS

- Solo se usan en subconsultas.
- Producen resultado true/false.
- True si y solo si existe al menos un renglón al menos en la tabla regresada por la subconsulta.
- False si la subconsulta regresa una tabla vacía.
- NOT EXISTS es lo opuesto de EXISTS.

# EXISTS and NOT EXISTS

- Dado que (NOT) EXISTS checa solo existencia y no existencia de renglones en la tabla de resultante de una subconsulta, la subconsulta puede tener cualquier número de columnas.

- Estas subconsultas comúnmente siguen la forma:

```
(SELECT * ...)
```

# Example 5.31 Query using EXISTS

Encontrar a todo el personal que trabaja en una sucursal de London.

```
SELECT staffNo, fName, lName, position
   FROM Staff s
   WHERE EXISTS  ( SELECT *
                   FROM Branch b
                   WHERE s.branchNo = b.branchNo AND
                         city = ‘London’);
```

# Example 5.31 Query using EXISTS

**Table 5.31** Result table for Example 5.31.

| staffNo | fName | lName | position |
|---|---|---|---|
| SL21 | John | White | Manager |
| SL41 | Julie | Lee | Assistant |

# Example 5.31 Query using EXISTS

- Note, search condition s.branchNo = b.branchNo is necessary to consider correct branch record for each member of staff.
- If omitted, would get all staff records listed out because subquery:

```
SELECT * FROM Branch WHERE city='London'
```

- would always be true and query would be:

```
SELECT staffNo, fName, lName, position FROM Staff
WHERE true;
```

# Example 5.31 Query using EXISTS

- Could also write this query using join construct:

```
SELECT staffNo, fName, lName, position
   FROM Staff s, Branch b
   WHERE s.branchNo = b.branchNo AND    city = ‘London’;
```

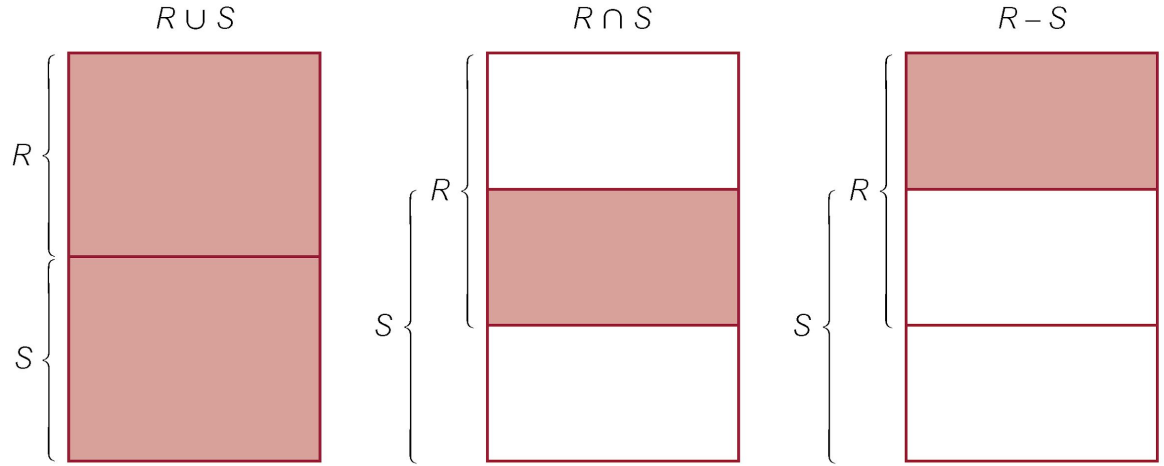
# Union, Intersect, and Difference (Except)

- Combinar resultados de dos o más consultas en una sola tabla.
- La *unión* de dos tablas, A y B, es una tabla que contiene todas las tuplas en A o B o en ambas.
- La *intersección* es una tabla que contiene las tuplas comunes de A y B.
- La *diferencia* es una tabla que contiene todas las tuplas en A, pero no en B.
- Las tablas tienen que ser compatibles con la unión.

# Union, Intersect, and Difference (Except)

- Format of set operator clause in each case is:
  - *op* [ALL] [CORRESPONDING [BY {column1 [, ...]}]]
- If CORRESPONDING BY specified, set operation performed on the named column(s).
- If CORRESPONDING specified but not BY clause, operation performed on common columns.
- If ALL specified, result can include duplicate rows.

# Union, Intersect, and Difference (Except)




(a) Union (b) Intersection (c) Difference

# Example 5.32 Use of UNION

List all cities where there is either a branch office or a property.

```
(SELECT city
 FROM Branch
 WHERE city IS NOT NULL) UNION
     (SELECT city
         FROM PropertyForRent
             WHERE city IS NOT NULL);
```

# Example 5.32 Use of UNION

- Or

```
(SELECT *
    FROM Branch
    WHERE city IS NOT NULL) UNION CORRESPONDING BY city
        (SELECT *
            FROM PropertyForRent
            WHERE city IS NOT NULL);
```

# Example 5.32 Use of UNION

- Produces result tables from both queries and merges both tables together.

**Table 5.32** Result table for Example 5.32.

| city |
| --- |
| London |
| Glasgow |
| Aberdeen |
| Bristol |

# Example 5.33 Use of INTERSECT

List all cities where there is both a branch office and a property.

```
(SELECT city
   FROM Branch) INTERSECT (SELECT city
                           FROM PropertyForRent);
```

# Example 5.33 Use of INTERSECT

- Or

```
(SELECT *
     FROM Branch) INTERSECT CORRESPONDING BY city
          (SELECT *
               FROM PropertyForRent);
```

**Table 5.33** Result table for Example 5.33.

| city |
| --- |
| Aberdeen |
| Glasgow |
| London |

# Example 5.33 Use of INTERSECT

- Could rewrite this query without INTERSECT operator:

```
SELECT b.city
       FROM Branch b PropertyForRent p
       WHERE b.city = p.city;
```

- Or:

```
SELECT DISTINCT city
        FROM Branch b
       WHERE EXISTS
       (SELECT *
            FROM PropertyForRent p
            WHERE p.city = b.city);
```

# Example 5.34 Use of EXCEPT

List of all cities where there is a branch office but no properties.

```
(SELECT city FROM Branch)
    EXCEPT
       (SELECT city FROM PropertyForRent);
```

Or

```
(SELECT * FROM Branch)
    EXCEPT CORRESPONDING BY city
       (SELECT * FROM PropertyForRent);
```

# Example 5.34 Use of EXCEPT

Could rewrite this query without EXCEPT:

```
SELECT DISTINCT city FROM Branch
WHERE city NOT IN
  (SELECT city FROM PropertyForRent);
```

Or

```
SELECT DISTINCT city FROM Branch b
WHERE NOT EXISTS
  (SELECT * FROM PropertyForRent p
  WHERE p.city = b.city);
```

# INSERT

```
INSERT INTO TableName [ (columnList) ]
VALUES (dataValueList)
```

- *columnList* es opcional; si se omite, SQL asume la lista de todas las columnas en su orden CREATE TABLE original.
- Cualquier columna omitida debe haber sido declarada como NULL cuando se creó la tabla, a menos que se haya especificado DEFAULT al crear la columna.

# INSERT

- *dataValueList* debe coincidir con columnList de la siguiente manera:
  - el número de elementos de cada lista debe ser el mismo;
  - debe haber correspondencia directa en la posición de los elementos en dos listas;
  - El tipo de datos de cada elemento en *dataValueList* debe ser compatible con el tipo de datos de la columna correspondiente.

# Example 5.35 INSERT … VALUES

Inserte una nueva fila en la tabla Staff proporcionando datos para todas las columnas.

```
INSERT INTO Staff
   VALUES   (‘SG16’,   ‘Alan’,   ‘Brown’,   ‘Assistant’,   ‘M’,
      Date‘1957-05-25’, 8300, ‘B003’);
```

# Example 5.36 INSERT using Defaults

Inserte una nueva fila en la tabla Staff proporcionando datos para todas las columnas obligatorias.

```
INSERT INTO Staff (staffNo, fName, lName, position, salary,
   branchNo)
   VALUES                                    (‘SG44’,‘Anne’,‘Jones’,
      ‘Assistant’, 8100, ‘B003’);
```

- Or

```
INSERT INTO Staff VALUES (‘SG44’, ‘Anne’, ‘Jones’, ‘Assistant’,
NULL,  NULL, 8100, ‘B003’);
```

# UPDATE

```
UPDATE TableName
SET columnName1 = dataValue1
     [, columnName2 = dataValue2...]
[WHERE searchCondition]
```

- *TableName* puede ser el nombre de una tabla base o una vista actualizable.
- La cláusula SET especifica los nombres de una o más columnas que se actualizarán.

# UPDATE

- La cláusula WHERE es opcional:
  - si se omite, las columnas con nombre se actualizan para todas las filas de la tabla;
  - si se especifica, solo se actualizan las filas que satisfacen la condición de búsqueda.
- Los nuevos *dataValue* deben ser compatibles con el tipo de datos de la columna correspondiente.

# Example 5.38/39 UPDATE All Rows

Dar a todo el personal un aumento salarial del 3%.

```
UPDATE Staff
 SET salary = salary*1.03;
```

Dar a todos los gerentes un aumento salarial del 5%.

```
UPDATE Staff
   SET salary = salary*1.05
   WHERE position = ‘Manager’;
```

# Example 5.40 UPDATE Multiple Columns

Ascienda a David Ford (staffNo = "SG14") a Gerente y cambie su salario a £18,000.

```
UPDATE Staff
    SET position = ‘Manager’, salary = 18000
    WHERE staffNo = ‘SG14’;
```

# DELETE

```
DELETE FROM TableName
[WHERE searchCondition]
```

- *TableName* puede ser el nombre de una tabla base o una vista actualizable.
- *searchCondition* es opcional; si se omite, todas las filas se eliminan de la tabla. Esto no elimina la tabla. Si se especifica *searchCondition*, solo se eliminan las filas que cumplen la condición.

# Example 5.41/42 DELETE Specific Rows

Elimine todas las visitas relacionadas con la propiedad PG4.

```
DELETE FROM Viewing
    WHERE propertyNo = ‘PG4’;
```

Elimina todos los registros de la tabla de visualización.

```
DELETE FROM Viewing;
```