Structured Query Language (SQL)

Objectives of SQL

- Ideally, database language should allow user to:
  - create the database and relation structures;
  - perform insertion, modification, deletion of data from relations;
  - perform simple and complex queries.

- Must perform these tasks with minimal user effort and command structure and syntax must be easy to learn.

- It must be portable.

Objectives of SQL

- SQL is a transform-oriented language with 2 major components:
  - A DDL for defining the database structure.
  - A DML for retrieving and updating data.

- SQL does not contain flow control commands. These must be implemented using a programming or job-control language, or interactively by the decisions of the user.

Objectives of SQL

- SQL is relatively easy to learn:
  - It is a non-procedural language - you specify what information you require, rather than how to get it.

  - It is essentially free-format.
Objectives of SQL

- Consists of standard English words:

  ```sql
  CREATE TABLE staff(sno VARCHAR(5),
  lname VARCHAR(15),
  salary DECIMAL(7,2));
  INSERT INTO staff
  VALUES ('SG16', 'Brown', 8300);
  SELECT sno, lname, salary
  FROM staff
  WHERE salary > 10000;
  ```

Objectives of SQL

- Can be used by a range of users including DBAs, management, application programmers, and other types of end users.

- An ISO standard now exists for SQL, making it both the formal and de facto standard language for relational databases.

History of SQL

- In 1974, D. Chamberlin (IBM San Jose Laboratory) defined language called 'Structured English Query Language' or SEQUEL.

- In late 70s, ORACLE appeared and was probably first commercial RDBMS based on SQL.

- In 1987, ANSI and ISO published an initial standard for SQL.

- In 1992, first major revision to ISO standard occurred, referred to as SQL2 or SQL/92.

Importance of SQL

- SQL has become part of application architectures such as IBM's Systems Application Architecture (SAA).

- It is strategic choice of many large and influential organizations (e.g. X/OPEN).

- SQL is Federal Information Processing Standard (FIPS) to which conformance is required for all sales of databases to American Government.
Writing SQL Commands

- SQL statement consists of **reserved words** and **user-defined words**.
  - Reserved words are a fixed part of SQL and must be spelt exactly as required and cannot be split across lines.
  - User-defined words are made up by user and represent names of various database objects such as relations, columns, views.

Writing SQL Commands

- Most components of an SQL statement are **case insensitive**, except for literal character data.
- More readable with indentation and lineation:
  - Each clause should begin on a new line.
  - Start of a clause should line up with start of other clauses.
  - If clause has several parts, should each appear on a separate line and be indented under start of clause.

SELECT Statement

```
SELECT [DISTINCT | ALL] 
(* | [column_expression [AS new_name]] [,...] ) 
FROM table_name [alias] [, ...]
[WHERE condition]
[GROUP BY column_list]
[HAVING condition]
[ORDER BY column_list]
```

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

- Order of the clauses cannot be changed.
- Only SELECT and FROM are mandatory.
- Example:
  
  List full details of all staff:
  ```sql
  SELECT sno, fname, lname, address, tel_no, position, sex, dob, salary, nin, bno
  FROM staff;
  ```

- Can use * as an abbreviation for 'all columns':
  ```sql
  SELECT *
  FROM staff;
  ```

---

**Example 1  All Columns, All Rows**

```sql
SELECT *
FROM staff;
```

**Example 2 Specific columns, all rows**

```sql
SELECT sno, fname, lname, salary
FROM staff;
```

---

**Example 3 Use of DISTINCT**

Use DISTINCT to eliminate duplicates:

```sql
SELECT pno
FROM viewing;
SELECT DISTINCT pno
FROM viewing;
```

---

CS3462 Introduction to Database Systems
Helena Wong, 2001
Example 4 Calculated Fields

```
SELECT sno, fname, lname, salary/12
FROM staff;
```

<table>
<thead>
<tr>
<th>sno</th>
<th>fname</th>
<th>lname</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL21</td>
<td>John</td>
<td>White</td>
<td>2500.00</td>
</tr>
<tr>
<td>SG37</td>
<td>Ann</td>
<td>Beech</td>
<td>1600.00</td>
</tr>
<tr>
<td>SG14</td>
<td>David</td>
<td>Ford</td>
<td>1500.00</td>
</tr>
<tr>
<td>SA9</td>
<td>Mary</td>
<td>Howe</td>
<td>750.00</td>
</tr>
<tr>
<td>SG5</td>
<td>Susan</td>
<td>Brand</td>
<td>2000.00</td>
</tr>
<tr>
<td>SL41</td>
<td>Julie</td>
<td>Lee</td>
<td>750.00</td>
</tr>
</tbody>
</table>

(6 rows)

To name column, use AS clause:
```
SELECT sno, fname, lname, salary/12
AS monthly_salary
FROM staff;
```

Example 5 Comparison Search Condition

```
SELECT sno, fname, lname, position, salary
FROM staff
WHERE salary > 10000;
```

<table>
<thead>
<tr>
<th>sno</th>
<th>fname</th>
<th>lname</th>
<th>position</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL21</td>
<td>John</td>
<td>White</td>
<td>Manager</td>
<td>30000.00</td>
</tr>
<tr>
<td>SG37</td>
<td>Ann</td>
<td>Beech</td>
<td>Sr Asst</td>
<td>12000.00</td>
</tr>
<tr>
<td>SG14</td>
<td>David</td>
<td>Ford</td>
<td>Deputy</td>
<td>18000.00</td>
</tr>
<tr>
<td>SG5</td>
<td>Susan</td>
<td>Brand</td>
<td>Manager</td>
<td>24000.00</td>
</tr>
</tbody>
</table>

(4 rows)

Example 6 Compound Comparison Search Condition

```
SELECT bno, street, area, city, pcode
FROM branch
WHERE city = 'London' OR city = 'Glasgow';
```

<table>
<thead>
<tr>
<th>bno</th>
<th>street</th>
<th>area</th>
<th>city</th>
<th>pcode</th>
</tr>
</thead>
<tbody>
<tr>
<td>B5</td>
<td>22 Deer Rd</td>
<td>Sldcp</td>
<td>London</td>
<td>SW1 4EJ</td>
</tr>
<tr>
<td>B3</td>
<td>163 Main St</td>
<td>Partick</td>
<td>Glasgow</td>
<td>G11 9QX</td>
</tr>
<tr>
<td>B2</td>
<td>56 Clover Dr</td>
<td>London</td>
<td></td>
<td>NW10 6EU</td>
</tr>
</tbody>
</table>

(3 rows)

Example 7 Range Search Condition

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

```
SELECT sno, fname, lname, position, salary
FROM staff
WHERE salary BETWEEN 20000 AND 30000;
```

<table>
<thead>
<tr>
<th>sno</th>
<th>fname</th>
<th>lname</th>
<th>position</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL21</td>
<td>John</td>
<td>White</td>
<td>Manager</td>
<td>30000.00</td>
</tr>
<tr>
<td>SG5</td>
<td>Susan</td>
<td>Brand</td>
<td>Manager</td>
<td>24000.00</td>
</tr>
</tbody>
</table>

(2 rows)
Example 7 Range Search Condition

- Also a negated version NOT BETWEEN.
- BETWEEN does not add much to SQL's expressive power. Could also write:

```sql
SELECT sno, fname, lname, position, salary
FROM staff
WHERE salary >= 20000 AND salary <= 30000;
```

- Useful, though, for a range of values.

Example 8 Set Membership

SELECT sno, fname, lname, position
FROM staff
WHERE position IN ('Manager', 'Deputy');

<table>
<thead>
<tr>
<th>sno</th>
<th>fname</th>
<th>lname</th>
<th>position</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL21</td>
<td>John</td>
<td>White</td>
<td>Manager</td>
</tr>
<tr>
<td>SG14</td>
<td>David</td>
<td>Ford</td>
<td>Deputy</td>
</tr>
<tr>
<td>SG5</td>
<td>Susan</td>
<td>Brand</td>
<td>Manager</td>
</tr>
</tbody>
</table>

(3 rows)

Example 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:

```sql
SELECT sno, fname, lname, position
FROM staff
WHERE position='Manager' OR position='Deputy';
```

- IN is more efficient when set contains many values.

Example 9 Pattern Matching

SELECT sno, fname, lname, address, salary
FROM staff
WHERE address LIKE '%Glasgow%';

<table>
<thead>
<tr>
<th>sno</th>
<th>fname</th>
<th>lname</th>
<th>address</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>SG37</td>
<td>Ann</td>
<td>Beech</td>
<td>81 George St, Glasgow PA1 2JR</td>
<td>12000.00</td>
</tr>
<tr>
<td>SG14</td>
<td>David</td>
<td>Ford</td>
<td>63 Ashby St, Patrick, Glasgow G11</td>
<td>18000.00</td>
</tr>
<tr>
<td>SG5</td>
<td>Susan</td>
<td>Brand</td>
<td>5 Ctr Western Rd, Glasgow G12</td>
<td>24000.00</td>
</tr>
</tbody>
</table>

(3 rows)
Example 9 Pattern Matching

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

Example 10 NULL Search Condition

There are 2 viewings for property PG4, one with and one without a comment. Using special keyword IS NULL:

```
SELECT rno, date
FROM viewing
WHERE pno = 'PG4' AND comment IS NULL;
```

```
rno   date
CR56   26-May-98
(1 row)
```

Negated version (IS NOT NULL) can test for non-null values.

Example 11 Single Column Ordering

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

```
SELECT sno, fname, lname, salary
FROM staff
ORDER BY salary DESC;
```

```
sno  fname  lname  salary
SL21  John   White  30000.00
SG5  Susan  Brandt  24000.00
SG14  David  Ford  18000.00
SG37  Ann    Beech  12000.00
SA9  Mary   Howe  9000.00
SL41  Julie  Lee  9000.00
(6 rows)
```

Example 12 Multiple Column Ordering

```
SELECT pno, type, rooms, rent
FROM property_for_rent
ORDER BY type;
```

```
pno  type  rooms  rent
PL94  Flat  4     400
PG4  Flat  3     350
PG16  Flat  4     450
PA14  House 6     650
PG21  House 5     600
(6 rows)
```
ISO standard defines five aggregate functions:

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

Each operates on a single column of a table and return single value.

- COUNT, MIN, and MAX apply to numeric and non-numeric fields, but SUM and AVG may be used on numeric fields only.
- Apart from COUNT(*), each function eliminates nulls first and operates only on remaining non-null values.

**COUNT(\*)** counts all rows of a table, regardless of whether nulls or duplicate values occur.

- Can use DISTINCT before column name to eliminate duplicates.
- DISTINCT has no effect with MIN/MAX, but may have with SUM/AVG.
- Aggregate functions can be used only in SELECT list and in HAVING clause.

If SELECT list includes an aggregate function and there is no GROUP BY clause, then SELECT list cannot reference a column outwith an aggregate function. For example, following is illegal:

```sql
SELECT sno, COUNT(salary)
FROM staff;
```
Example 13  Use of COUNT(*)

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

SELECT COUNT(*) AS count
FROM property_for_rent
WHERE rent > 350;

<table>
<thead>
<tr>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
</tr>
</tbody>
</table>

(1 row)

Example 14  Use of COUNT(DISTINCT)

How many different properties viewed in May ‘98?

SELECT COUNT(DISTINCT pno) AS count
FROM viewing
WHERE date BETWEEN DATE'1998-05-01'
AND DATE'1998-05-31';

<table>
<thead>
<tr>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
</tr>
</tbody>
</table>

(1 row)

Example 15  Use of COUNT and SUM

Find number of Managers and sum of their salaries.

SELECT COUNT(sno) AS count,
      SUM(salary) AS sum
FROM staff
WHERE position = 'Manager';

<table>
<thead>
<tr>
<th>count</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>54000.00</td>
</tr>
</tbody>
</table>

(1 row)

Example 16  Use of MIN, MAX, AVG

Find minimum, maximum, and average staff salary.

SELECT MIN(salary) AS min,
      MAX(salary) AS max,
      AVG(salary) AS avg
FROM staff;

<table>
<thead>
<tr>
<th>count</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>54000.00</td>
</tr>
</tbody>
</table>

(1 row)
SELECT Statement - Grouping

- Use GROUP BY clause to get sub-totals.
- SELECT and GROUP BY closely integrated: each item in SELECT list must be single-valued per group, and SELECT clause may only contain:
  - Column names.
  - Aggregate functions.
  - Constants.
  - An expression involving combinations of the above.

Example 17 Use of GROUP BY

```
SELECT bno, COUNT(sno) AS count, SUM(salary) AS sum
FROM staff
GROUP BY bno
ORDER BY bno;
```

<table>
<thead>
<tr>
<th>bno</th>
<th>count</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>B3</td>
<td>3</td>
<td>54000.00</td>
</tr>
<tr>
<td>B5</td>
<td>2</td>
<td>39000.00</td>
</tr>
<tr>
<td>B7</td>
<td>1</td>
<td>9000.00</td>
</tr>
</tbody>
</table>

(3 rows)

SELECT Statement - Grouping

- All column names in SELECT list must appear in GROUP BY clause unless name is used only in an aggregate function.
- If WHERE is used with GROUP BY, WHERE is applied first, then groups are formed from remaining rows satisfying predicate.
- ISO considers two nulls to be equal for purposes of GROUP BY.

Restricted Grouping

- HAVING clause is designed for use with GROUP BY clause to restrict groups that appear in final result table.
- Similar to WHERE, but WHERE filters individual rows whereas HAVING filters groups.
- Column names in HAVING clause must also appear in the GROUP BY list or be contained within an aggregate function.
Example 18 Use of HAVING

```
SELECT bno, COUNT(sno) AS count, SUM(salary) AS sum
FROM staff
GROUP BY bno
HAVING COUNT(sno) > 1
ORDER BY bno;
```

<table>
<thead>
<tr>
<th>bno</th>
<th>count</th>
<th>sum</th>
</tr>
</thead>
<tbody>
<tr>
<td>B3</td>
<td>3</td>
<td>54000.00</td>
</tr>
<tr>
<td>B5</td>
<td>2</td>
<td>39000.00</td>
</tr>
</tbody>
</table>

(2 rows)

Subqueries

- Some SQL statements can have a SELECT embedded within them.
- A subselect can be used in WHERE and HAVING clauses of an outer SELECT, where it is called a subquery or nested query.
- Subselects may also appear in INSERT, UPDATE, and DELETEs.

Example 19 Subquery with Equality

```
SELECT sno, fname, lname, position
FROM staff
WHERE bno =
  (SELECT bno
   FROM branch
   WHERE street = '163 Main St');
```

<table>
<thead>
<tr>
<th>sno</th>
<th>fname</th>
<th>lname</th>
<th>position</th>
</tr>
</thead>
<tbody>
<tr>
<td>SG37</td>
<td>Ann</td>
<td>Beech</td>
<td>Sr Asst</td>
</tr>
<tr>
<td>SG14</td>
<td>David</td>
<td>Ford</td>
<td>Deputy</td>
</tr>
<tr>
<td>SG5</td>
<td>Susan</td>
<td>Brand</td>
<td>Manager</td>
</tr>
</tbody>
</table>

(3 rows)

Example 20 Subquery with Aggregate

```
SELECT sno, fname, lname, position, salary
FROM staff
WHERE salary >
  (SELECT avg(salary)
   FROM staff);
```

- Cannot write "WHERE salary > avg(salary)".

<table>
<thead>
<tr>
<th>sno</th>
<th>fname</th>
<th>lname</th>
<th>position</th>
<th>salary_diff</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL21</td>
<td>John</td>
<td>White</td>
<td>Manager</td>
<td>13000.00</td>
</tr>
<tr>
<td>SG14</td>
<td>David</td>
<td>Ford</td>
<td>Deputy</td>
<td>10000.00</td>
</tr>
<tr>
<td>SG5</td>
<td>Susan</td>
<td>Brand</td>
<td>Manager</td>
<td>7000.00</td>
</tr>
</tbody>
</table>

(3 rows)
**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 21  Nested subquery: use of IN**

```
SELECT pno, street, area, city, pcode, type, rooms, rent
FROM property_for_rent
WHERE sno IN
  (SELECT sno
   FROM staff
   WHERE bno =
     (SELECT bno
      FROM branch
      WHERE street = '163 Main St'));
```

**ANY and ALL**

- ANY and ALL may be used with subqueries that produce a single column of numbers.
- If subquery preceded by ALL, condition will only be true if it is satisfied by all values produced by subquery.
- If subquery preceded by 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.
- ISO standard allows SOME to be used in place of ANY.

**Example 22  Use of ANY/SOME**

```
SELECT sno, fname, lname, position, salary
FROM staff
WHERE salary > SOME
  (SELECT salary
   FROM staff
   WHERE bno = 'B3');
```

<table>
<thead>
<tr>
<th>sno</th>
<th>fname</th>
<th>lname</th>
<th>position</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL21</td>
<td>John</td>
<td>White</td>
<td>Manager</td>
<td>30000.00</td>
</tr>
<tr>
<td>SG14</td>
<td>David</td>
<td>Ford</td>
<td>Deputy</td>
<td>18000.00</td>
</tr>
<tr>
<td>SG5</td>
<td>Susan</td>
<td>Brand</td>
<td>Manager</td>
<td>24000.00</td>
</tr>
</tbody>
</table>

(3 rows)
**Example 23 Use of ALL**

```sql
SELECT sno, fname, lname, position, salary
FROM staff
WHERE salary > ALL
    (SELECT salary
     FROM staff
     WHERE bno = 'B3');
```

<table>
<thead>
<tr>
<th>sno</th>
<th>fname</th>
<th>lname</th>
<th>position</th>
<th>salary</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL21</td>
<td>John</td>
<td>White</td>
<td>Manager</td>
<td>30000.00</td>
</tr>
</tbody>
</table>

(1 row)

**Multi-Table Queries**

- Can use subqueries provided result columns come from same table.
- If result columns come from more than one table must use a join.
- To perform join, include more than one table in FROM clause.
- Use comma as separator and typically include WHERE clause to specify join column(s).
- Also possible to use an alias for a table named in FROM clause.
- Alias is separated from table name with a space.
- Alias can be used to qualify column names when there is ambiguity.

**Example 24 Simple Join**

```sql
SELECT r.rno, fname, lname, pno, comment
FROM renter r, viewing v
WHERE r.rno = v.rno;
```

- Equivalent to equi-join in relational algebra.

<table>
<thead>
<tr>
<th>rno</th>
<th>fname</th>
<th>lname</th>
<th>pno</th>
<th>comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>CR56</td>
<td>Aline</td>
<td>Stewart</td>
<td>PG36</td>
<td></td>
</tr>
<tr>
<td>CR56</td>
<td>Aline</td>
<td>Stewart</td>
<td>PA14</td>
<td>too small</td>
</tr>
<tr>
<td>CR56</td>
<td>Aline</td>
<td>Stewart</td>
<td>PG4</td>
<td></td>
</tr>
<tr>
<td>CR62</td>
<td>Mary</td>
<td>Tregear</td>
<td>PA14</td>
<td>no dining room</td>
</tr>
<tr>
<td>CR76</td>
<td>John</td>
<td>Kay</td>
<td>PG4</td>
<td>too remote</td>
</tr>
</tbody>
</table>

(5 rows)

**Alternative JOIN Constructs**

- SQL2 provides alternative ways to specify joins:
  - FROM renter r JOIN viewing v ON r.rno = v.rno
  - FROM renter JOIN viewing USING rno
  - FROM renter NATURAL JOIN viewing

- In each case, FROM replaces original FROM and WHERE. However, first one produces table with two identical Rno columns, remaining two produce table with single Rno column.
Example 25 Sorting a join

SELECT s.bno, s.sno, fname, lname, pno
FROM staff s, property_for_rent p
WHERE s.sno = p.sno
ORDER BY s.bno, s.sno, pno;

<table>
<thead>
<tr>
<th>bno</th>
<th>sno</th>
<th>fname</th>
<th>lname</th>
<th>pno</th>
</tr>
</thead>
<tbody>
<tr>
<td>B3</td>
<td>SG14</td>
<td>David</td>
<td>Ford</td>
<td>PG16</td>
</tr>
<tr>
<td>B3</td>
<td>SG14</td>
<td>David</td>
<td>Ford</td>
<td>PG4</td>
</tr>
<tr>
<td>B3</td>
<td>SG37</td>
<td>Ann</td>
<td>Beech</td>
<td>PG21</td>
</tr>
<tr>
<td>B3</td>
<td>SG37</td>
<td>Ann</td>
<td>Beech</td>
<td>PG36</td>
</tr>
<tr>
<td>B5</td>
<td>SL41</td>
<td>Julie</td>
<td>Lee</td>
<td>PL94</td>
</tr>
<tr>
<td>B7</td>
<td>SA9</td>
<td>Mary</td>
<td>Howe</td>
<td>PA14</td>
</tr>
</tbody>
</table>

(6 rows)

Example 26 Three Table Join

SELECT b.bno, b.city, s.sno, fname, lname, pno
FROM branch b, staff s, property_for_rent p
WHERE b.bno = s.bno AND s.sno = p.sno
ORDER BY b.bno, s.sno, pno;

<table>
<thead>
<tr>
<th>bno</th>
<th>city</th>
<th>sno</th>
<th>fname</th>
<th>lname</th>
<th>pno</th>
</tr>
</thead>
<tbody>
<tr>
<td>B3</td>
<td>Glasgow</td>
<td>SG14</td>
<td>David</td>
<td>Ford</td>
<td>PG16</td>
</tr>
<tr>
<td>B3</td>
<td>Glasgow</td>
<td>SG14</td>
<td>David</td>
<td>Ford</td>
<td>PG4</td>
</tr>
<tr>
<td>B3</td>
<td>Glasgow</td>
<td>SG37</td>
<td>Ann</td>
<td>Beech</td>
<td>PG21</td>
</tr>
<tr>
<td>B3</td>
<td>Glasgow</td>
<td>SG37</td>
<td>Ann</td>
<td>Beech</td>
<td>PG36</td>
</tr>
<tr>
<td>B5</td>
<td>London</td>
<td>SL41</td>
<td>Julie</td>
<td>Lee</td>
<td>PL94</td>
</tr>
<tr>
<td>B7</td>
<td>Aberdeen</td>
<td>SA9</td>
<td>Mary</td>
<td>Howe</td>
<td>PA14</td>
</tr>
</tbody>
</table>

(6 rows)

Example 26 Three Table Join

SQL2 provides alternative formulations for FROM and WHERE:
FROM (branch b JOIN staff s USING bno) AS bs
JOIN property_for_rent p USING sno

Example 27 Multiple Grouping Columns

Find number of properties handled by each staff member in each branch.
SELECT s.bno, s.sno, COUNT(*) AS count
FROM staff s, property_for_rent p
WHERE s.sno = p.sno
GROUP BY s.bno, s.sno
ORDER BY s.bno, s.sno;

<table>
<thead>
<tr>
<th>bno</th>
<th>sno</th>
<th>count</th>
</tr>
</thead>
<tbody>
<tr>
<td>B3</td>
<td>SG14</td>
<td>2</td>
</tr>
<tr>
<td>B3</td>
<td>SG37</td>
<td>2</td>
</tr>
<tr>
<td>B5</td>
<td>SL41</td>
<td>1</td>
</tr>
<tr>
<td>B7</td>
<td>SA9</td>
<td>1</td>
</tr>
</tbody>
</table>

(4 rows)
### Computing a Join

Procedure for generating results of join are:
1. Form Cartesian product of the tables named in FROM clause.
2. If there is a WHERE clause, apply the search condition to each row of the product table, retaining those rows that satisfy the condition.
3. For each remaining row, determine the value of each item in the SELECT list to produce a single row in the result table.
4. If SELECT DISTINCT has been specified, eliminate any duplicate rows from the result table.
5. If there is an ORDER BY clause, sort the result table as required.

### Outer Joins

- With a join, if one row of a table is unmatched, row is omitted from result table.
- The outer join operations retain rows that do not satisfy the join condition.
- Consider following two simplified tables:

<table>
<thead>
<tr>
<th>BRANCH1</th>
<th>PROPERTY_FOR_RENT1</th>
</tr>
</thead>
<tbody>
<tr>
<td>bno</td>
<td>city</td>
</tr>
<tr>
<td>B3</td>
<td>Glasgow</td>
</tr>
<tr>
<td>B4</td>
<td>Bristol</td>
</tr>
<tr>
<td>B2</td>
<td>London</td>
</tr>
</tbody>
</table>

#### Example 28 Left Outer Join

```sql
SELECT b.*, p.*
FROM branch1 b LEFT JOIN property_for_rent1 p ON b.bcity = p.pcity;
```

- Includes those rows of first (left) table unmatched with rows from second (right) table.
- Columns from second table are filled with NULLs.
Example 29  Right Outer Join

```sql
SELECT b.*, p.*
FROM branch1 b RIGHT JOIN
    property_for_rent1 p ON b.bcity = p.pcity;
```

- Right outer join includes those rows of 2nd (right) table that are unmatched with rows from 1st (left) table.
- Columns from first table are filled with NULLs.

<table>
<thead>
<tr>
<th>bno</th>
<th>bcity</th>
<th>pno</th>
<th>pcity</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>PA14</td>
<td>Aberdeen</td>
</tr>
<tr>
<td>B3</td>
<td>Glasgow</td>
<td>PG4</td>
<td>Glasgow</td>
</tr>
<tr>
<td>B2</td>
<td>London</td>
<td>PL94</td>
<td>London</td>
</tr>
</tbody>
</table>

(3 rows)

Example 30  Full Outer Join

```sql
SELECT b.*, p.*
FROM branch1 b FULL JOIN
    property_for_rent1 p ON b.bcity = p.pcity;
```

- Includes those rows that are unmatched in both tables.
- Unmatched columns are filled with NULLs.

<table>
<thead>
<tr>
<th>bno</th>
<th>bcity</th>
<th>pno</th>
<th>pcity</th>
</tr>
</thead>
<tbody>
<tr>
<td>NULL</td>
<td>NULL</td>
<td>PA14</td>
<td>Aberdeen</td>
</tr>
<tr>
<td>B3</td>
<td>Glasgow</td>
<td>PG4</td>
<td>Glasgow</td>
</tr>
<tr>
<td>B4</td>
<td>Bristol</td>
<td>NULL</td>
<td>NULL</td>
</tr>
<tr>
<td>B2</td>
<td>London</td>
<td>PL94</td>
<td>London</td>
</tr>
</tbody>
</table>

(4 rows)

EXISTS and NOT EXISTS

- EXISTS and NOT EXISTS are for use only with subqueries, producing true/false result.
- Since EXISTS and NOT EXISTS check only for existence or non-existence of rows in subquery result table, subquery can contain any number of columns.
- Common for subqueries following (NOT) EXISTS to be of form:
  ```sql
  (SELECT * ...)  
  ```

Example 31  Query using EXISTS

```sql
SELECT sno, fname, lname, position
FROM staff s
WHERE EXISTS
  (SELECT *
   FROM branch b
   WHERE s.bno = b.bno AND city = 'London');
```

<table>
<thead>
<tr>
<th>sno</th>
<th>fname</th>
<th>lname</th>
<th>position</th>
</tr>
</thead>
<tbody>
<tr>
<td>SL21</td>
<td>John</td>
<td>White</td>
<td>Manager</td>
</tr>
<tr>
<td>SL41</td>
<td>Julie</td>
<td>Lee</td>
<td>Assistant</td>
</tr>
</tbody>
</table>

(2 rows)
Union, Intersect, and Difference (Except)

- Can use normal set operations of union, intersection, and difference to combine results of two or more queries into a single result table.
- Union of two tables, A and B, is table containing all rows in either A or B or both.
- Intersection is table containing all rows common to both A and B.
- Difference is table containing all rows in A but not in B.
- Two tables must be union compatible.

Format of set operator clause in each case is:

\[ \text{op [ALL] [CORRESPONDING [BY \{ column1 [, \ldots \} ] ]]} \]

- 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.

Example 32  Use of UNION

\[
\begin{align*}
\text{(SELECT area} & \quad \text{FROM branch} \\
\text{WHERE area IS NOT NULL)} & \quad \text{UNION} \\
\text{(SELECT area} & \quad \text{FROM property_for_rent} \\
\text{WHERE area IS NOT NULL);} \\
\text{Or} & \\
\text{(SELECT *} & \quad \text{FROM branch} \\
\text{WHERE area IS NOT NULL)} & \quad \text{UNION CORRESPONDING BY area} \\
\text{(SELECT *} & \quad \text{FROM property_for_rent} \\
\text{WHERE area IS NOT NULL);} \\
\end{align*}
\]

<table>
<thead>
<tr>
<th>area</th>
<th>Sidcup</th>
<th>Dyce</th>
<th>Partick</th>
<th>Leigh</th>
<th>Dee</th>
<th>Kilburn</th>
<th>Hyndland</th>
</tr>
</thead>
<tbody>
<tr>
<td>(7 rows)</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example 33 Use of INTERSECT

\[(\text{SELECT city FROM branch}) \ \text{INTERSECT} \ \text{(SELECT city FROM property\_for\_rent)};\]

Or

\[(\text{SELECT * FROM branch}) \ \text{INTERSECT CORRESPONDING BY city} \ \text{(SELECT * FROM property\_for\_rent)};\]

\[\text{SELECT city FROM branch b property\_for\_rent p} \ \text{WHERE b.city = p.city;}\]

Example 34 Use of EXCEPT

\[\text{SELECT city FROM branch) EXCEPT (SELECT city FROM property\_for\_rent);}\]

Or

\[(\text{SELECT * FROM branch}) \ \text{EXCEPT CORRESPONDING BY city} \ \text{(SELECT * FROM property\_for\_rent);}\]

<table>
<thead>
<tr>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>Aberdeen</td>
</tr>
<tr>
<td>Glasgow</td>
</tr>
<tr>
<td>London</td>
</tr>
</tbody>
</table>

(3 rows)

<table>
<thead>
<tr>
<th>city</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bristol</td>
</tr>
</tbody>
</table>

(1 row)

INSERT (Format I)

\[\text{INSERT INTO table\_name \[ (column\_list) \]} \ \text{VALUES (data\_value\_list)}\]

- \(\text{column\_list}\) is optional.
- If omitted, SQL assumes a list of all columns in their original CREATE TABLE order.
- Any columns omitted must have been declared as NULL when table was created, unless DEFAULT was specified when creating column.

INSERT

- \(\text{data\_value\_list}\) must match \(\text{column\_list}\) as follows:
- Number of items in each list must be the same.
- Must be direct correspondence in position of items in two lists.
- Data type of each item in \(\text{data\_value\_list}\) must be compatible with data type of corresponding column.
Example 35  INSERT ... VALUES

INSERT INTO staff
VALUES ('SG16', 'Alan', 'Brown',
'67 Endrick Rd, Glasgow G32 8QX',
'0141-211-3001', 'Assistant', 'M', '25-May-57',
8300, 'WN848391H', 'B3');

Example 36  INSERT using Defaults

INSERT INTO staff (sno, fname, lname, position,
salary, bno)
VALUES ('SG44', 'Anne', 'Jones', 'Assistant',
8100, 'B3');

Or

INSERT INTO staff
VALUES ('SG44', 'Anne', 'Jones', NULL, NULL,
'Assistant', NULL, NULL, 8100, NULL, 'B3');

Example 37  INSERT ... SELECT

```
INSERT INTO staff_prop_count
(SELECT s.sno, fname, lname, COUNT(*)
FROM staff s, property_for_rent p
WHERE s.sno = p.sno
GROUP BY s.sno, fname, lname)
UNION
(SELECT sno, fname, lname, 0
FROM staff
WHERE sno NOT IN
(SELECT DISTINCT sno
FROM property_for_rent));
```
**UPDATE**

`UPDATE table_name
SET column_name1 = data_value1
[, column_name2 = data_value2...]
[WHERE search_condition]`

- `table_name` can be name of a base table or an updatable view.
- SET clause specifies names of one or more columns that are to be updated.

**Example 38  UPDATE All Rows**

Give all staff a 3% pay increase.

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

**Example 39  UPDATE Specific Rows**

- Give all Managers a 5% pay increase.
  ```
  UPDATE staff
  SET salary = salary*1.05
  WHERE position = 'Manager';
  ```
  - WHERE clause finds rows that contain data for Managers. Update is applied only to these particular rows.
Example 40  UPDATE Multiple Columns

Promote David Ford (sno = 'SG14') to Manager and change his salary to 18,000.

UPDATE staff
SET position = 'Manager', salary = 18000
WHERE sno = 'SG14';

Example 41  DELETE Specific Rows

Delete all viewings that relate to property PG4.

DELETE FROM viewing
WHERE pno = 'PG4';

DELETE

DELETE FROM table_name
[WHERE search_condition]

- table_name can be name of a base table or an updatable view.
- search_condition is optional; if omitted, all rows are deleted from table. This does not delete table. If search_condition is specified, only those rows that satisfy condition are deleted.

Example 42  DELETE All Rows

Delete all records from the Viewing table.

DELETE FROM viewing;
ISO SQL Data Types

<table>
<thead>
<tr>
<th>Data type</th>
<th>Declarations</th>
</tr>
</thead>
<tbody>
<tr>
<td>character</td>
<td>CHAR, VARCHAR</td>
</tr>
<tr>
<td>bit</td>
<td>BIT, BIT VARYING</td>
</tr>
<tr>
<td>exact numeric</td>
<td>NUMERIC, DECIMAL, INTEGER, SMALLINT</td>
</tr>
<tr>
<td>approximate numeric</td>
<td>FLOAT, REAL, DOUBLE PRECISION</td>
</tr>
<tr>
<td>datetime</td>
<td>DATE, TIME, TIMESTAMP</td>
</tr>
<tr>
<td>interval</td>
<td>INTERVAL</td>
</tr>
</tbody>
</table>

Data Definition

- In SQL92, relations and other database objects exist in an environment.
- Each environment contains one or more catalogs, and each catalog consists of set of schemas.
- Schema is a named collection of related database objects.
- Objects in a schema can be tables, views, domains, assertions, collations, translations, and character sets. All have same owner.

CREATE SCHEMA [name | AUTHORIZATION creator_id ]
DROP SCHEMA name [RESTRICT | CASCADE ]

- With RESTRICT (default), schema must be empty or operation fails.
- With CASCADE, operation cascades to drop all objects associated with schema in the order defined above. If any of these operations fail, DROP SCHEMA fails.

CREATE TABLE (Basic)

- Creates a table with one or more columns of the specified data_type.
- NULL (default) indicates whether column can contain nulls.
- With NOT NULL, system rejects any attempt to insert a null in the column.
- Primary keys should always be specified as NOT NULL.
- Foreign keys are often (but not always) candidates for NOT NULL.
Example 43 CREATE TABLE

CREATE TABLE property_for_rent (pno VARCHAR(5) NOT NULL, street VARCHAR(25) NOT NULL, area VARCHAR(15), city VARCHAR(15) NOT NULL, pcode VARCHAR(8), type CHAR(1) NOT NULL, rooms SMALLINT NOT NULL, rent DECIMAL(6,2) NOT NULL, ono VARCHAR(5) NOT NULL, sno VARCHAR(5), bno VARCHAR(3) NOT NULL);

DROP TABLE

DROP TABLE tbl_name [RESTRICT | CASCADE]
e.g. DROP TABLE property_for_rent;

- Removes named table and all rows within it.
- With RESTRICT, if any other objects depend for their existence on continued existence of this table, SQL does not allow request.
- With CASCADE, SQL drops all dependent objects (and objects dependent on these cts).