BLG 317E
DATABASE SYSTEMS

WEEK 4

Slides credit:
S. Shivakumar, CS 145, Stanford University
Database System Concepts, Silberschatz, Korth, Sudarshan

ITU

Structured Query Language (SQL)
Selection-Projection-Join and Nested Queries

SQL SELECT Command

- SELECT command is used to retrieve (querying) data rows from tables.
- It does not change any data values in tables.
- The SELECT and FROM clauses are compulsory, others are optional.
- Clauses (components) of the query have particular execution phases (semantically).
- Actual execution phases are determined by DBMS.

General Syntax of SELECT Command

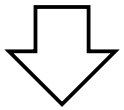
	Execution phases
SELECT <attribute list>	5th
FROM <table list>	1st
[WHERE <conditions>]	2nd
[GROUP BY <grouping attributes>]	3rd
[HAVING <group conditions>]	4th
[ORDER BY <attribute list>]	6th

SELECT Statement

Asterisk (*) returns all columns in the result-set.

Query

```
select *
from DEPARTMENT;
```



Result-set (table)

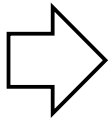
dept_id	dept_name	building
BIO	Biology	Watson Bld.
CS	Computer Science	Taylor Bld.
EE	Electronics Eng.	Taylor Bld.
FIN	Finance	Painter Bld.
HIS	History	Painter Bld.
MU	Music	Packard Bld.
PHY	Physics	Watson Bld.
SPR	Sports	NULL

SELECT Statement

The required columns can be specified in SELECT clause.

```
select dept_id, dept_name
from DEPARTMENT;
```

dept_id	dept_name	building
BIO	Biology	Watson Bld.
CS	Computer Science	Taylor Bld.
EE	Electronics Eng.	Taylor Bld.
FIN	Finance	Painter Bld.
HIS	History	Painter Bld.
MU	Music	Packard Bld.
PHY	Physics	Watson Bld.
SPR	Sports	NULL



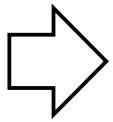
dept_id	dept_name
BIO	Biology
CS	Computer Science
EE	Electronics Eng.
FIN	Finance
HIS	History
MU	Music
PHY	Physics
SPR	Sports

LIMIT operator

- The LIMIT operator reduces the number of rows in result set.
- It returns the first **N** records.
- The default starting row number is 1.

```
select dept_name
from DEPARTMENT
limit 3;
```

dept_id	dept_name	building
BIO	Biology	Watson Bld.
CS	Computer Science	Taylor Bld.
EE	Electronics Eng.	Taylor Bld.
FIN	Finance	Painter Bld.
HIS	History	Painter Bld.
MU	Music	Packard Bld.
PHY	Physics	Watson Bld.
SPR	Sports	NULL



dept_name
Biology
Computer Science
Electronics Eng.

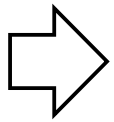
OFFSET operator

The OFFSET operator can be used with LIMIT operator.

Syntax : LIMIT N OFFSET K
Skips the first K rows, and returns the next N rows.

```
select dept_name
from DEPARTMENT
limit 3  OFFSET 4;
```

dept_id	dept_name	building
BIO	Biology	Watson Bld.
CS	Computer Science	Taylor Bld.
EE	Electronics Eng.	Taylor Bld.
FIN	Finance	Painter Bld.
HIS	History	Painter Bld.
MU	Music	Packard Bld.
PHY	Physics	Watson Bld.
SPR	Sports	NULL

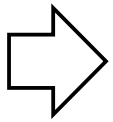


dept_name
History
Music
Physics

SELECT statement

- The following query means :
  “Find Department ID’s which has courses”.
- The result set contains duplicate rows.

```
select dept_id
from COURSE;
```



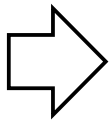
dept_id
BIO
BIO
BIO
CS
CS
CS
CS
CS
EE
EE
EE
FIN
HIS
MU
PHY
SPR

DISTINCT operator

dept_id
BIO
BIO
BIO
CS
CS
CS
CS
CS
EE
EE
EE
FIN
HIS
MU
PHY
SPR

The DISTINCT operator eliminates the duplicate rows in the result set.

```
select distinct dept_id
from COURSE;
```



dept_id
BIO
CS
EE
FIN
HIS
MU
PHY
SPR

Operators in WHERE Clause

- The WHERE clause is used to filter records (rows) in the result set.
- It is used to extract only those records that fulfill a specified condition.
- Condition may contain many operators.
- Usually WHERE clause is the most important clause in a SELECT statement.

Syntax

```
SELECT column1, column2, ...
FROM   table_name
WHERE  condition;
```

Operators in WHERE Clause

The following operators can be used in the WHERE clause conditions.

Operator	Description
=	Equal
>	Greater than
<	Less than
>=	Greater than or equal
<=	Less than or equal
<>, !=	Not equal
BETWEEN	Between a certain range
LIKE	Search for a string pattern
IN	Specifies multiple possible values for a column
IS NULL	Returns true if a field has no value
IS NOT NULL	Returns true if a field has a value

WHERE Clause

Query : Find all courses info that are offered by the Computer Science department (dept_id = 'CS').

```
select *
from COURSE
where dept_id = 'CS';
```

course_id	course_title	dept_id	credits
CS-101	Intro. to Algorithms	CS	4
CS-190	Web Design	CS	4
CS-315	Robotics	CS	3
CS-319	Image Processing	CS	3
CS-347	Database System Concepts	CS	3

WHERE Clause

Query : Find all courses info that are offered by departments other than the Computer Science department (dept_id <> 'CS').

```
select * from COURSE
where dept_id <> 'CS';
```

course_id	course_title	dept_id	credits
BIO-101	Intro. to Biology	BIO	4
BIO-301	Molecular Biology	BIO	4
BIO-401	Biology Lab	BIO	3
EE-184	Intro. to Digital Systems	EE	3
EE-208	Circuit Analysis	EE	5
EE-306	Semiconductors	EE	2
FIN-201	Accounting Methods	FIN	3
HIS-351	World History	HIS	3
MU-199	Music Production	MU	3
PHY-101	Optics Principles	PHY	4
SPR-202	Swimming	SPR	1

LIKE Operator

- The LIKE operator is used in WHERE clause to search for a specified string pattern in a column.
- There are two wildcards often used in conjunction with the LIKE operator.
  - The percent sign (%) represents zero, one, or multiple characters
  - The underscore sign (_) represents one, single character
- The percent sign and the underscore can also be used in combinations

Syntax

```
SELECT  column1, column2, ...
FROM    table_name
WHERE   column LIKE pattern;
```

LIKE Operator

Query : Find all courses info, whose **course_title** starts with 'Intro' word.

```
select * from COURSE
where course_title LIKE 'Intro%';
```

```
+-----------+---------------------------+---------+---------+
| course_id | course_title              | dept_id | credits |
+-----------+---------------------------+---------+---------+
| BIO-101   | Intro. to Biology         | BIO     |       4 |
| CS-101    | Intro. to Algorithms      | CS      |       4 |
| EE-184    | Intro. to Digital Systems | EE      |       3 |
+-----------+---------------------------+---------+---------+
```

LIKE Operator

Examples	Description
`WHERE column LIKE 'H%'`	Start with 'H'
`WHERE column LIKE '%o'`	End with 'o'
`WHERE column LIKE '_e%'`	Have 'e' in the second location
`WHERE column LIKE '%Hello%'`	Have 'Hello' in any location
`WHERE column LIKE 'H%o'`	Start with 'H' and ends with 'o'

UPDATE Statement

Query: Add 2 credits to the credits of all courses, whose **course_title** starts with 'Intro' word.

```
update COURSE
       set credits = credits + 2
where course_title LIKE 'Intro%';
```

ORDER BY Clause

- The ORDER BY keyword is used to sort the result-set in ascending or descending order.
- Default order is ascending (ASC) order by default.
- To sort the records in descending order, use the DESC keyword.

Syntax

```
SELECT column1, column2, ...
FROM table_name
ORDER BY column1  ASC | DESC,
         column2  ASC | DESC,
          ... ;
```

ORDER BY Clause

- **Query :** Find all student names and their department ID's.
- Sort the result set firstly by the dept_id field, then by the student_name field within the same department.
- Both sortings are in ascending order (default order).

```
select dept_id, student_name
from STUDENT
order by dept_id, student_name;
```

Result

dept_id	student_name
BIO	Bradley
CS	Brown
CS	Clark
CS	Patel
CS	Williams
EE	Blake
EE	Hudson
FIN	Chavez
HIS	Brown
MU	Sanchez
PHY	Snow
PHY	Walker
PHY	Young

Arithmetic Operations

The standard arithmetic operators + , - , * , /
can be applied to numeric values in an SQL query result.

Query : List the student names, their total credits, and the calculated numeric value of necessary credits to graduate.
(Formula = 120 – total_credits.)

```
select student_name,
       total_credits,
       (120 - total_credits)
from STUDENT;
```

Arithmetic Operations

```
select student_name,
       total_credits,
       (120 - total_credits)
from STUDENT;
```

Result

student_name	total_credits	(120 - total_credits)
Patel	32	88
Clark	102	18
Brown	80	40
Chavez	110	10
Young	56	64
Walker	46	74
Williams	54	66
Sanchez	38	82
Snow	0	120
Brown	58	62
Hudson	60	60
Blake	98	22
Bradley	120	0

Using Aliases (temporary renamings) in SELECT

- Aliases are used to give a table, or a column in a table, a temporary name.
- Aliases are often used to make column names more readable.
- An alias only exists for the duration of that query.
- An alias is created with the AS keyword.

Column alias syntax

```
SELECT column_name AS alias_name

FROM table_name;
```

Table alias syntax

```
SELECT column_name(s)

FROM table_name AS alias_name;
```

Column alias

Query

```
select student_name,
       total_credits,
       (120 - total_credits) AS credits_left
from STUDENT;
```

Result

student_name	total_credits	credits_left
Patel	32	88
Clark	102	18
Brown	80	40
Chavez	110	10
Young	56	64
Walker	46	74
Williams	54	66
Sanchez	38	82
Snow	0	120
Brown	58	62
Hudson	60	60
Blake	98	22
Bradley	120	0

AND, OR, NOT Operators

- The WHERE clause can be combined with AND, OR, and NOT operators.
- The AND and OR operators are used to filter records based on more than one condition.

Syntax 1

```
SELECT column1, column2, ...

FROM table_name

WHERE condition1 AND
      condition2 AND
      condition3 ...;
```

Syntax 2

```
SELECT column1, column2, ...

FROM table_name

WHERE NOT condition;
```

OR Operator

Query : Find all students info, who are students of either History, Music, or Finance department.

```
select *
from STUDENT
where dept_id = 'HIS' OR
      dept_id = 'MU' OR
      dept_id = 'FIN';
```

Result

student_id	student_name	dept_id	total_credits
23121	Chavez	FIN	110
19991	Brown	HIS	80
55739	Sanchez	MU	38

IN Operator

The IN operator is a shorthand for multiple OR conditions.

Syntax

```
SELECT column_name(s)
FROM   table_name
WHERE  column_name
       IN (value1, value2, ...);
```

IN Operator

Query : The same query as the previous.
"Find all students info, who are students of either History, Music, or Finance department".

```
select * from STUDENT
where dept_id IN ('HIS', 'MU', 'FIN');
```

Result (same)

student_id	student_name	dept_id	total_credits
23121	Chavez	FIN	110
19991	Brown	HIS	80
55739	Sanchez	MU	38

MULTI-TABLE QUERIES - JOINS

Table Join

- **Join:** Merging data from multiple tables into a single set of data (result-set).
- It is performed by matching tuples (records) of two relations (tables) over common values of one or more attributes (columns).
- The join operations do not modify the tables.
- The joined result is only a temporary result record set (output table) returned by the SELECT query.

Table Joins in WHERE clause (Equi-Join / Implicit Inner Join)

Syntax

```
SELECT  columns
FROM    table1, table2, table3, ...
WHERE   table1.key1 = table2.key1 AND
        table2.key2 = table3.key2 AND ...
```

The equality condition operator (=) is Equi-Join operator.

Table Join

Query : Find the names of all students, with the names of their departments.

```
select student_name, dept_name
from STUDENT, DEPARTMENT
where student.dept_id = department.dept_id;
```

Result

student_name	dept_name
Patel	Computer Science
Clark	Computer Science
Brown	History
Chavez	Finance
Young	Physics
Walker	Physics
Williams	Computer Science
Sanchez	Music
Snow	Physics
Brown	Computer Science
Hudson	Electronics Eng.
Blake	Electronics Eng.
Bradley	Biology

SELECT Statement Execution Plan

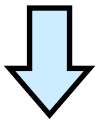
The following plan is used by DBMS, for the execution of given SQL query. DBMS uses two-nested loops.

SQL Query

```
select student_name, dept_name

from STUDENT, DEPARTMENT

where student.dept_id = department.dept_id;
```



Execution plan (pseudocode)

```
for each row in STUDENT

    for each row in DEPARTMENT

        if (STUDENT.dept_id == DEPARTMENT.dept_id)

            output (STUDENT.student_name,
                    DEPARTMENT.dept_name)
```

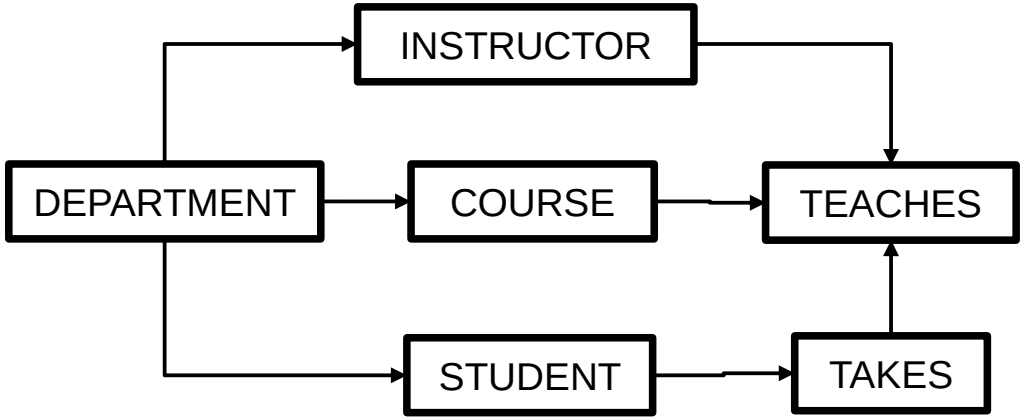
Table Join

Query : Find all department names, instructor names for each department, and course titles taught by each instructor. The result record set will be sorted by department name, instructor name, and course title.

```
select dept_name, instructor_name, course_title
from DEPARTMENT, INSTRUCTOR, TEACHES, COURSE
where
    department.dept_id        =  instructor.dept_id     AND
    instructor.instructor_id  =  teaches.instructor_id  AND
    teaches.course_id         =  course.course_id
order by dept_name, instructor_name, course_title;
```

- The comparison of **department.dept_id = instructor.dept_id** is an implicit inner join operation between DEPARTMENT table and INSTRUCTOR table.
- The “=” comparison operator is used as a join operator (equi-join).

Database Diagram



Result of query

dept_name	instructor_name	course_title
Biology	Thompson	Intro. to Biology
Biology	Thompson	Molecular Biology
Computer Science	Fixter	Image Processing
Computer Science	Fixter	Web Design
Computer Science	Hernandez	Image Processing
Computer Science	Hernandez	Intro. to Algorithms
Computer Science	Murphy	Database System Concepts
Computer Science	Murphy	Intro. to Algorithms
Computer Science	Murphy	Robotics
Electronics Eng.	Smith	Intro. to Digital Systems
Finance	Marsh	Accounting Methods
History	Castillo	World History
Music	Jones	Music Production
Physics	Smith	Optics Principles

Table Join

Query : Find all department names, instructor names for each department, and course titles taught by each instructor. Exclude the departments whose dept_id is either 'CS' or 'EE'.

```
select dept_name, instructor_name, course_title
from DEPARTMENT, INSTRUCTOR, TEACHES, COURSE
where department.dept_id  =  instructor.dept_id  AND
      instructor.instructor_id  =  teaches.instructor_id AND
      teaches.course_id  =  course.course_id AND
      NOT (department.dept_id = 'CS'  OR  department.dept_id = 'EE')
order by dept_name, instructor_name, course_title;
```

Result

dept_name	instructor_name	course_title
Biology	Thompson	Intro. to Biology
Biology	Thompson	Molecular Biology
Finance	Marsh	Accounting Methods
History	Castillo	World History
Music	Jones	Music Production
Physics	Smith	Optics Principles

Table Join

Query : List all courses (id and title) and the instructor names who taught those courses.

```
select course.course_id, course_title, instructor_name
from COURSE, INSTRUCTOR, TEACHES
where course.course_id  =  teaches.course_id  AND
      teaches.instructor_id  =  instructor.instructor_id
order by course_id, instructor_name;
```

Result

course_id	course_title	instructor_name
BIO-101	Intro. to Biology	Thompson
BIO-301	Molecular Biology	Thompson
CS-101	Intro. to Algorithms	Hernandez
CS-101	Intro. to Algorithms	Murphy
CS-190	Web Design	Fixter
CS-315	Robotics	Murphy
CS-319	Image Processing	Fixter
CS-319	Image Processing	Hernandez
CS-347	Database System Concepts	Murphy
EE-184	Intro. to Digital Systems	Smith
FIN-201	Accounting Methods	Marsh
HIS-351	World History	Castillo
MU-199	Music Production	Jones
PHY-101	Optics Principles	Smith

Table Join

Query : Find all student names, course titles that were taken by students, with the grades for each course taken. Sort the results by student name.

```
select student_name, course_title, grade
from STUDENT, TAKES, TEACHES, COURSE
where student.student_id = takes.student_id
  AND takes.CRN          = teaches.CRN
  AND teaches.course_id  = course.course_id
order by student_name;
```

Result

student_name	course_title	grade
Blake	Accounting Methods	C-
Blake	Optics Principles	B
Bradley	Intro. to Algorithms	A
Bradley	Robotics	B
Brown	Accounting Methods	A
Brown	World History	A
Brown	Web Design	B
Chavez	Molecular Biology	C+
Clark	Database System Concepts	A
Clark	Image Processing	A-
Hudson	Intro. to Biology	C
Patel	Database System Concepts	C
Patel	Music Production	A
Patel	Optics Principles	A
Patel	Image Processing	A
Sanchez	Image Processing	A-
Walker	Database System Concepts	C
Walker	World History	B
Williams	Accounting Methods	A-
Williams	Music Production	B+
Young	Intro. to Digital Systems	B-

Table aliases

Query

```
select student_name, course_title, grade
from STUDENT s, TAKES t, TEACHES te, COURSE c
where s.student_id = t.student_id
  AND t.CRN        = te.CRN
  AND te.course_id = c.course_id
order by student_name;
```

Result

student_name	course_title	grade
Blake	Accounting Methods	C-
Blake	Optics Principles	B
Bradley	Intro. to Algorithms	A
Bradley	Robotics	B
Brown	Accounting Methods	A
Brown	World History	A
Brown	Web Design	B
Chavez	Molecular Biology	C+
Clark	Database System Concepts	A
Clark	Image Processing	A-
Hudson	Intro. to Biology	C
Patel	Database System Concepts	C
Patel	Music Production	A
Patel	Optics Principles	A
Patel	Image Processing	A
Sanchez	Image Processing	A-
Walker	Database System Concepts	C
Walker	World History	B
Williams	Accounting Methods	A-
Williams	Music Production	B+
Young	Intro. to Digital Systems	B-

SUBQUERIES

Subqueries

- A subquery is a query that is nested inside another query.
- Subqueries can be interpreted as single values or as whole relations.
- A relation can be:
  - Used as input for another query
  - Checked for containment of a value
- A subquery can return a Single-Value result-set, or Multiple-Values result set.
- A subquery is used in WHERE clause mostly.
- Also it can be used in SELECT clause, or in FROM clause.

Subqueries

Syntax

```
SELECT   Columns1
FROM     Tables1
WHERE    Condition1
         (SELECT   Columns2
          FROM     Tables2
          WHERE    Condition2)
```

Outer query (main query)

Subquery (inner query / nested query)

Result-set of Subquery	Operators used in **Condition1**
Single-value	=, <, <=, >, >=
Multiple-values	EXISTS, IN, ANY, ALL

Single Value Subquery

- The simplest subquery returns exactly one column and exactly one row as the result-set (1x1 table).
- It can be used with comparison operators **=, <, <=, >, >=.**

Query.Part1 : Find the minimum price value in SALES table. No subquery is used. Result has single value.

SALES table

sale_id	product_name	price	quantity	month
101	Orange	3.0	22	Jan
102	Kiwi	0.8	34	Jan
103	Orange	1.5	23	Feb
104	Apple	2.0	38	Feb
105	Cherry	2.7	44	Feb
106	Apricot	1.2	78	Feb
107	Orange	3.0	76	March
108	Cherry	4.1	60	March
109	Apple	1.6	48	March

```
select MIN(price)
from SALES;
```

```
+------------+
| MIN(price) |
+------------+
|        0.8 |
+------------+
```

Query.Part2 : Find all records whose price field are equal to the minimum price in table.

```
select *
from SALES
where price  =
    (select MIN(price)
     from SALES);
```

```
+---------+--------------+-------+----------+-------+
| sale_id | product_name | price | quantity | month |
+---------+--------------+-------+----------+-------+
|     102 | Kiwi         |   0.8 |       34 | Jan   |
+---------+--------------+-------+----------+-------+
```

Single Value Subquery (= operator)

Query : Find the records whose quantity are equal to the maximum quantity in 'February'.

```
select *
from SALES
where quantity  =
      (select MAX(quantity)
       from SALES
       where Month='Feb');
```

```
+---------+--------------+-------+----------+-------+
| sale_id | product_name | price | quantity | month |
+---------+--------------+-------+----------+-------+
|     106 | Apricot      |   1.2 |       78 | Feb   |
+---------+--------------+-------+----------+-------+
```

Single Value Subquery (>= operator)

Query : Find all records whose prices are **greater than** or equal to the average price in table.

```
select *
from SALES
where price  >=
      (select AVG(price)
       from SALES);
```

sale_id	product_name	price	quantity	month
101	Orange	3.0	22	Jan
105	Cherry	2.7	44	Feb
107	Orange	3.0	76	March
108	Cherry	4.1	60	March

Query.Part1 : Get a list of all records from SALES table. Each row in result-set should also contain the **Average Price** information.

```
select *,
       (select AVG(price)
        from SALES) AS avg_price

from SALES;
```

sale_id	product_name	price	quantity	month	avg_price
101	Orange	3.0	22	Jan	2.21111
102	Kiwi	0.8	34	Jan	2.21111
103	Orange	1.5	23	Feb	2.21111
104	Apple	2.0	38	Feb	2.21111
105	Cherry	2.7	44	Feb	2.21111
106	Apricot	1.2	78	Feb	2.21111
107	Orange	3.0	76	March	2.21111
108	Cherry	4.1	60	March	2.21111
109	Apple	1.6	48	March	2.21111

Query.Part2 : Find all records whose price field are **less than the average** price in SALES table. (Filtered version of **Query.Part1**)

```
select *, (select AVG(price) from SALES)
          AS avg_price
from SALES
where price  <
      (select AVG(price) from SALES) ;
```

```
+---------+--------------+-------+----------+-------+----------+
| sale_id | product_name | price | quantity | month |avg_price|
+---------+--------------+-------+----------+-------+----------+
|     102 | Kiwi         |   0.8 |       34 | Jan   |  2.21111 |
|     103 | Orange       |   1.5 |       23 | Feb   |  2.21111 |
|     104 | Apple        |   2.0 |       38 | Feb   |  2.21111 |
|     106 | Apricot      |   1.2 |       78 | Feb   |  2.21111 |
|     109 | Apple        |   1.6 |       48 | March |  2.21111 |
+---------+--------------+-------+----------+-------+----------+
```

Multiple Values Subquery

- A subquery can also return multiple columns or multiple rows.
- It can be used with operators **EXISTS, IN, ANY, ALL.**
- Optionally, the **NOT** operator can also be used before the EXISTS and IN operators.

Operator	Description
EXISTS	It is used to test for the existence of any record in a subquery. It returns TRUE if the subquery returns one or more records.
IN	It allows to specify multiple values in a WHERE clause. The IN operator is a shorthand for multiple OR conditions.
ANY	Returns boolean value TRUE as a result, if ANY of the subquery values meet the condition. ANY means that the condition will be true if the operation is true for any of the values in the range.
ALL	Returns TRUE if ALL of the subquery values meet the condition. It is used with SELECT, WHERE and HAVING statements. ALL means that the condition will be true only if the operation is true for all values in the range.

Multiple Values Subquery (EXISTS)

Syntax

```
SELECT column_names
FROM table_name
WHERE EXISTS
   (SELECT column_name
    FROM table_name
    WHERE condition);
```

Multiple Values Subquery (EXISTS)

```
SELECT student_name
FROM student s
WHERE EXISTS
  (SELECT *
   FROM takes t
   WHERE t.student_id = s.student_id);
```

student_name
Emma Johnson
Noah Wilson
Olivia Davis
Liam Smith
Ava Brown
Sophia Evans
Mason Taylor
Ethan Thomas
Isabella Jackson
Oliver Miller
Charlotte Hall
James Adams
Amelia White
Benjamin Scott
Harper Green
Evelyn Moore
Samuel Roberts
Sophie Turner
Lucas Harris
Lily Clark
Jackson Thomas
Scarlett Lewis
Daniel Walker
Chloe Foster

Multiple Values Subquery (EXISTS)

```
SELECT student_name
FROM student s
WHERE NOT EXISTS
  (SELECT *
   FROM takes t
   WHERE t.student_id = s.student_id);
```

```
+------------------+
| student_name     |
+------------------+
| Ali Cakmak       |
+------------------+
```

Multiple Values Subquery (IN)

Syntax

```
SELECT column_names
FROM table_name
WHERE column_name  IN
  (SELECT column_name
   FROM table_name
   WHERE condition);
```

Multiple Values Subquery (IN)

Query : Find product names (other than 'Cherry') and their months, which were sold in the same months that the product 'Cherry' was sold.

```
SELECT sale_id, Product_name, Month
FROM SALES
WHERE Product_name <> 'Cherry' AND
   Month   IN
   ( SELECT Month
     FROM SALES
     WHERE Product_name = 'Cherry'
   );
```

sale_id	Product_name	Month
103	Orange	Feb
104	Apple	Feb
106	Apricot	Feb
107	Orange	March
109	Apple	March

Multiple Values Subquery (ANY/ALL)

Syntax

```
SELECT column_names
FROM table_name
WHERE column_name comparison_operator [ANY | ALL]
  (SELECT column_name
   FROM table_name
   WHERE condition);
```

Comparison Operators

=	<>	!=	>	>=	<	<=

Definition of "ALL" Clause

- F <comp> **all** $r \Leftrightarrow \forall\ t \in r$ (F <comp> *t)*

(5 < **all** | 0, 5, 6) = false

(5 < **all** | 6, 10) = true

(5 = **all** | 4, 5) = false

(5 ≠ **all** | 4, 6) = true (since 5 ≠ 4 and 5 ≠ 6)

(≠ **all**) ≡ **not in**
However, (= **all**) ≢ **in**

Definition of "ANY" Clause

- F <comp> **any** $r \Leftrightarrow \exists\, t \in r$ such that (F <comp> t)
  Where <comp> can be: $<,\ \leq,\ >,\ =,\ \neq$

(5 < **any** | 0 / 5 / 6 |) = true (read: 5 < some tuple in the relation)

(5 < **any** | 0 / 5 |) = false

(5 = **any** | 0 / 5 |) = true

(5 ≠ **any** | 0 / 5 |) = true (since 0 ≠ 5)

(= **any**) ≡ **in**
However, (≠ **any**) ≢ **not in**

Multiple Values Subquery (ALL operator)

Query : Find products that are more expensive than **ALL** 'Apple' products.

```
SELECT *
FROM SALES
WHERE Price > ALL
    ( SELECT Price
      FROM SALES
      WHERE Product_name = 'Apple'
    )
;
```

sale_id	product_name	price	quantity	month
101	Orange	3.0	22	Jan
105	Cherry	2.7	44	Feb
107	Orange	3.0	76	March
108	Cherry	4.1	60	March

Correlated Subquery

- A correlated subquery refers to the tables introduced in the outer query.
- It is a subquery that references an attribute from the outer query.
- A correlated subquery depends on the outer query.
- It cannot be run independently from the outer query.

Correlated Subquery (IN operator)

Query : Find product names (other than 'Cherry') and their months, which were sold on the same months that the product 'Cherry' was sold.

```
SELECT sale_id, Product_name, Month
FROM   SALES   other_sale
WHERE Month  IN
   ( SELECT Month
     FROM SALES
     WHERE Product_name = 'Cherry' AND
           other_sale.Product_name <> 'Cherry'
   );
```

sale_id	Product_name	Month
103	Orange	Feb
104	Apple	Feb
106	Apricot	Feb
107	Orange	March
109	Apple	March

Correlated Subquery

```
SELECT student_name
FROM student s
WHERE EXISTS
  (SELECT *
   FROM takes t
   WHERE t.student_id = s.student_id);
```

Equivalent queries!

```
SELECT student_name
FROM student s
WHERE s.student_id IN
  (SELECT student_id
   FROM takes);
```

```
+------------------+
| student_name     |
+------------------+
| Emma Johnson     |
| Noah Wilson      |
| Olivia Davis     |
| Liam Smith       |
| Ava Brown        |
| Sophia Evans     |
| Mason Taylor     |
| Ethan Thomas     |
| Isabella Jackson |
| Oliver Miller    |
| Charlotte Hall   |
| James Adams      |
| Amelia White     |
| Benjamin Scott   |
| Harper Green     |
| Evelyn Moore     |
| Samuel Roberts   |
| Sophie Turner    |
| Lucas Harris     |
| Lily Clark       |
| Jackson Thomas   |
| Scarlett Lewis   |
| Daniel Walker    |
| Chloe Foster     |
+------------------+
```

Subquery in SELECT and FROM clauses

- A subquery in SELECT clause must return a single value.
- A subquery in FROM clause may return a single value, or multiple values (table).
- In general, subqueries execute very slowly, compared to other methods.
- Subqueries should be avoided for performance reasons, if there is another faster solution such as joins.

Subquery in SELECT clause

Query : Find the sum of quantities for each group of products in the SALES table. Sort the result set by sum of quantities, in descending order.

Solution) Correlated Subquery

```
SELECT DISTINCT P1.product_name,
       (SELECT SUM(P2.quantity)
        FROM SALES AS P2
        WHERE P1.product_name = P2.product_name
        ) AS total
FROM SALES  AS P1
ORDER BY TOPLAM  DESC
```

```
+--------------+--------+
| product_name |  total |
+--------------+--------+
| Orange       |    121 |
| Cherry       |    104 |
| Apple        |     86 |
| Apricot      |     78 |
| Kiwi         |     34 |
+--------------+--------+
```

Subquery in FROM clause

Query : Find all records whose price field are equal to the minimum price in table.

Solution1) Subquery in **FROM** clause

```
select P1.*
from SALES AS P1,
     (select MIN(price) AS cheapest
      from SALES) AS P2
where P1.price = P2.cheapest ;
```

```
+---------+--------------+-------+----------+-------+
| sale_id | product_name | price | quantity | month |
+---------+--------------+-------+----------+-------+
|     102 | Kiwi         |   0.8 |       34 | Jan   |
+---------+--------------+-------+----------+-------+
```

Solution2) Subquery in **WHERE** clause.

```
select *
from SALES
where price  =
    (select MIN(price)
     from SALES);
```

Same result-set

```
+---------+--------------+-------+----------+-------+
| sale_id | product_name | price | quantity | month |
+---------+--------------+-------+----------+-------+
|     102 | Kiwi         |   0.8 |       34 | Jan   |
+---------+--------------+-------+----------+-------+
```

COMMON TABLE EXPRESSION

Common Table Expression (CTE) (The WITH clause)

- Common Table Expression (CTE) is an alternative way of writing a subquery, that helps simplify a main query.
- A CTE generates a temporary virtual table (with records and columns), during the execution of a query.
- It is a temporary named result-set created from a simple SELECT statement, that can be used in a subsequent SQL statement.
- It is like a named subquery, whose result is stored in a virtual table to be referenced later in the main que

Common Table Expression (CTE)

The WITH clause should be written before the main SQL statement.

Syntax

```
WITH cte_name AS
(
  SELECT ...
)

SELECT columns
FROM   tables
WHERE  condition
```

CTE (subquery)
(It is part of the main query)

Main query

Query : Get a list of all records from SALES table.
Each row in result-set should also contain the **Average Price** information.

```
WITH RESULT AS (
  SELECT AVG(Price) AS avg_price
  FROM SALES
)
SELECT *
FROM SALES, RESULT;
```

sale_id	product_name	price	quantity	month	avg_price
101	Orange	3.0	22	Jan	2.21111
102	Kiwi	0.8	34	Jan	2.21111
103	Orange	1.5	23	Feb	2.21111
104	Apple	2.0	38	Feb	2.21111
105	Cherry	2.7	44	Feb	2.21111
106	Apricot	1.2	78	Feb	2.21111
107	Orange	3.0	76	March	2.21111
108	Cherry	4.1	60	March	2.21111
109	Apple	1.6	48	March	2.21111

CASE Statement in SELECT clause

- The CASE statement can be used as part of the SELECT clause.
- It goes through conditions and returns a value, when the first condition is met.
- It is similar to an if-then-else , or switch statement in C language.
- Once a condition is true, it will stop reading and return the result.
- If no conditions are true, it returns the value in the ELSE clause.
- If there is no ELSE part and no conditions are true, it returns NULL.

Syntax

```
CASE
    WHEN condition1 THEN result1
    WHEN condition2 THEN result2
    WHEN conditionN THEN resultN
    ELSE result
END
```

CASE Statement in SELECT clause

Query : Find all student names, course titles that were taken by students, the grades for each course taken, and also a text explanation of grades.

```
select student_name, course_title, grade,
       CASE
          WHEN grade = 'A'  THEN 'Excellent'
          WHEN grade = 'A-' THEN 'Very good'
          WHEN grade = 'B+' THEN 'Good high'
          WHEN grade = 'B'  THEN 'Good medium'
          WHEN grade = 'B-' THEN 'Good low'
          WHEN grade = 'C+' THEN 'Passed high'
          WHEN grade = 'C'  THEN 'Passed medium'
          WHEN grade = 'C-' THEN 'Passed low'
          ELSE 'Failed'
       END
       AS evaluation

from STUDENT, TAKES, TEACHES, COURSE

where student.student_id  =  takes.student_id  AND
      takes.CRN           =  teaches.CRN       AND
      teaches.course_id   =  course.course_id
order by student_name;
```

Result of query

student_name	course_title	grade	evaluation
Blake	Accounting Methods	C-	Passed low
Blake	Optics Principles	B	Good medium
Bradley	Intro. to Algorithms	A	Excellent
Bradley	Robotics	B	Good medium
Brown	Accounting Methods	A	Excellent
Brown	World History	A	Excellent
Brown	Web Design	B	Good medium
Chavez	Molecular Biology	C+	Passed high
Clark	Database System Concepts	A	Excellent
Clark	Image Processing	A-	Very good
Hudson	Intro. to Biology	C	Passed medium
Patel	Database System Concepts	C	Passed medium
Patel	Music Production	A	Excellent
Patel	Optics Principles	A	Excellent
Patel	Image Processing	A	Excellent
Sanchez	Image Processing	A-	Very good
Walker	Database System Concepts	C	Passed medium
Walker	World History	B	Good medium
Williams	Accounting Methods	A-	Very good
Williams	Music Production	B+	Good high
Young	Intro. to Digital Systems	B-	Good low

Using INSERT command and SELECT command

- In the following example, the SELECT query replaces the VALUES keyword of the INSERT INTO command.
- The data from the result-set of SELECT command are added to the new table called student_course.
- In general, this method is used to build **temporary tables** only.

```
create table student_course
      (student_name  varchar(20),
       course_title  varchar(30),
       grade         varchar(2));
```

Using INSERT command and SELECT command

```
insert into student_course
select student_name, course_title, grade
from STUDENT, TAKES, TEACHES, COURSE
where student.student_id  = takes.student_id AND
      takes.CRN           = teaches.CRN AND
      teaches.course_id   = course.course_id
order by student_name;
```

The statement below drops (deletes) the temporary table from database.

```
DROP TABLE  student_course;
```

Table student_course

student_name	course_title	grade
Blake	Accounting Methods	C-
Blake	Optics Principles	B
Bradley	Intro. to Algorithms	A
Bradley	Robotics	B
Brown	Accounting Methods	A
Brown	World History	A
Brown	Web Design	B
Chavez	Molecular Biology	C+
Clark	Database System Concepts	A
Clark	Image Processing	A-
Hudson	Intro. to Biology	C
Patel	Database System Concepts	C
Patel	Music Production	A
Patel	Optics Principles	A
Patel	Image Processing	A
Sanchez	Image Processing	A-
Walker	Database System Concepts	C
Walker	World History	B
Williams	Accounting Methods	A-
Williams	Music Production	B+
Young	Intro. to Digital Systems	B-

NULL VALUES IN SQL

NULLs in SQL

- Whenever we don't have a value, we can put a NULL
- Can mean many things:
  - Value does not exists
  - Value exists but is unknown
  - Value not applicable
  - Etc.
- The schema specifies for each attribute if it can be null (*nullable* attribute) or not
- How does SQL cope with tables that have NULLs?

NULLs in SQL

- *For numerical operations*, NULL -> NULL:
  - If x = NULL then 4*(3-x)/7 is still NULL
- *For boolean operations*, in SQL there are three values:

FALSE	**=**	**0**
UNKNOWN	**=**	**0.5**
TRUE	**=**	**1**

  - If x= NULL then x="Joe" is UNKNOWN

NULLs in SQL

- C1 AND C2 = min(C1, C2)
- C1 OR C2 = max(C1, C2)
- NOT C1 = 1 – C1

```
SELECT *
FROM   Person
WHERE (age < 25)
  AND (height > 6 AND weight > 190)
```

Won't return e.g. (age=20 height=NULL weight=200)!

Rule in SQL: include only tuples that yield TRUE (1.0)

NULLs in SQL

Unexpected behavior:

```
SELECT *
FROM   Person
WHERE  age < 25 OR age >= 25
```

Some Persons are not included !

NULLs in SQL

Can test for NULL explicitly:

- x IS NULL
- x IS NOT NULL

```
SELECT *
FROM   Person
WHERE  age < 25 OR age >= 25
   OR age IS NULL
```

Now it includes all Persons!

NULLs in SQL

Query : Find departments which have no building information.

```
select * from DEPARTMENT
where building IS NULL;
```

```
+---------+-----------+----------+
| dept_id | dept_name | building |
+---------+-----------+----------+
| SPR     | Sports    | NULL     |
+---------+-----------+----------+
```

SELECT Query Exercises (Using example UNIVERSITY database)

1) Find students who has taken the course titled “Optics Principles”.
   Display student names, grades, and Year/Semester information.

2) Find names of instructors who has taught the same course.
   Display instructor names, course ID, and course titles information.

3) Find students who has taken any courses that was offered
   by a department other than his/her own department.

4) Find the courses that exist in the COURSE table,
   but do not exist in the TEACHES table.

5) Determine the class for each student, based on total credits of student.
   Use the following ranges for determining the class.
   Display student names and their class information.

Total Credits is Between	Class of Student
0 and 30	'Freshman'
31 and 60	'Sophomore'
61 and 90	'Junior'
91 and 120	'Senior'