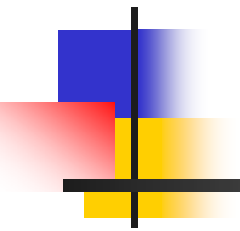
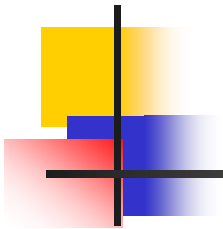
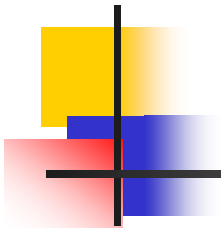


Structured Query Language - SQL



Objectives

- Example Tables +
- Introduction +
- ISO SQL Data Types +
- Comparison Operators in SQL +
- Logical Operators in SQL +
- Arithmetic Operators in SQL +
- SQL Schema and Catalog +
- SQL Data Definition Statements (DDL) +
- SQL Data Manipulation Statements (DML) +
- Other SQL Operators +



--- Example Table ...

City	Year	Cars_sold
Dhahran	2001	525
Dhahran	2002	456
Riyadh	2001	700
Riyadh	2002	654
Jeddah	2001	921
Jeddah	2002	752
Khobar	2002	

Car_Sales

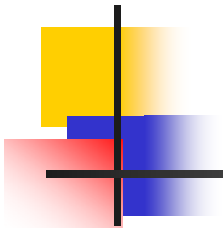
... --- Example: Tables

dno	dname
1	ICS
2	COE
3	SWE

Departments

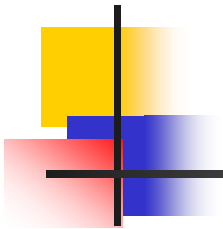
Lid	Lname	dno	salary
1	Ahmed	1	4000
2	Amin	2	3700
3	Hani	1	4200
4	Abdallah		4300
5	Ageel	1	4000
6	Yousef	2	3500
7	Khalid	2	4500

Lecturers



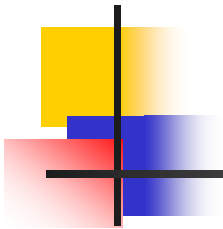
- Introduction

- Objectives of SQL +
- History of SQL +
- Importance of SQL +
- Components of SQL +
- Basic Guidelines for Writing SQL Statements +



-- 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.
- 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.
- 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.

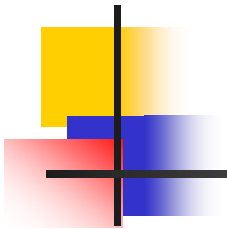
... -- Objectives of SQL

- Consists of standard English words:

```
CREATE TABLE staff(
                        sno      VARCHAR(5),
                        lname    VARCHAR(15),
                        salary   NUMBER(7,2)
                    );
```

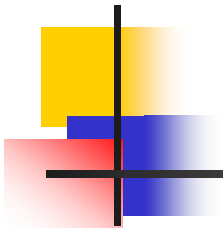
```
INSERT INTO staff
VALUES ('SG16', 'Brown', 8300);
```

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



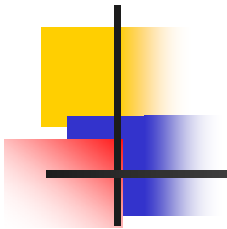
-- History of SQL ...

- In 1974, D. Chamberlin (IBM San Jose Laboratory) defined language called 'Structured English Query Language' or SEQUEL.
- A revised version SEQUEL/2 was defined in 1976 but name was subsequently changed to SQL for legal reasons.
- Still pronounced 'see-quel', though official pronunciation is 's-q-l'.
- IBM subsequently produced a prototype DBMS called *System R*, based on SEQUEL/2.
- Roots of SQL, however, are in SQUARE (Specifying Queries as Relational Expressions), which predates System R project.



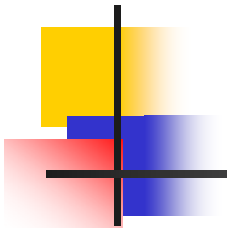
... -- History of SQL

- 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 1989, ISO published an addendum that defined an 'Integrity Enhancement Feature'.
- In 1992, first major revision to ISO standard occurred, referred to as SQL2 or SQL/92.



-- Importance of SQL ...

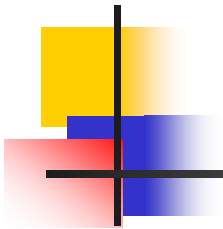
- SQL has become part of application architectures such as IBM's Systems Application Architecture (SAA).
- It is strategic choice of many large and influential organizations (e.g. X/OPEN).
- SQL is Federal Information Processing Standard (FIPS) to which conformance is required for all sales of databases to American Government.



... -- Importance of SQL

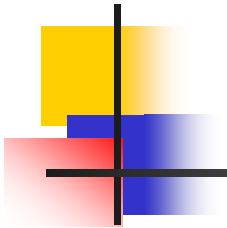
- SQL Access Group trying to define enhancements that will support interoperability across disparate systems.

- SQL is used in other standards and even influences development of other standards as a definitional tool. Examples include:
  - ISO's Information Resource Directory System (IRDS) Standard
  - Remote Data Access (RDA) Standard.



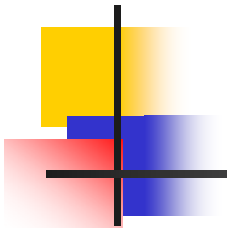
-- Components of SQL

- A database language must have support for the components listed below. Most implementations of SQL support various components listed below:
  - Data Definition Language (DDL)
  - Interactive Data Manipulation Language (Interactive DML)
  - Embedded Data Manipulation Language (Embedded DML)
  - Views
  - Integrity and transaction control
  - Authorization
  - Catalog and dictionary facility.



-- Basic Guidelines for Writing SQL Statements ...

- 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.
- 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.



... -- Basic Guidelines for Writing SQL Statements

- Use extended form of 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).

- ISO SQL Data Types

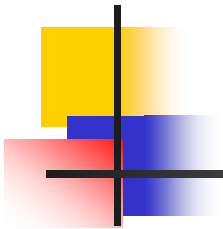
ISO SQL data types.

Data type	*Declarations*			
character	CHAR,	VARCHAR		
bit	BIT,	BIT VARYING		
exact numeric	NUMERIC,	DECIMAL,	INTEGER,	SMALLINT
approximate numeric	FLOAT,	REAL,	DOUBLE PRECISION	
datetime	DATE,	TIME,	TIMESTAMP	
interval	INTERVAL			

- Comparison Operators in SQL +

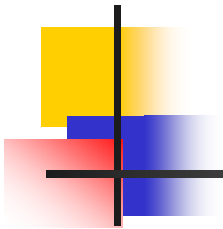
- There are six comparison operators in SQL. These operators are used to build conditions that are used in the WHERE clause of a DML statement:

Operator	Meaning
=	Equal
< >	Not Equal
<	Less than
>	Greater than
< =	Less than or Eqaul
> =	Greater than or Eqaul



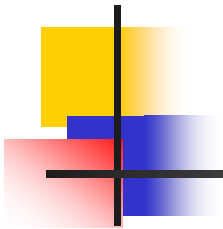
- Logical Operators in SQL

- There are three logical operators that help us to build compound conditions to be used in the WHERE clause of the SELECT statement.
  - The **AND** operator joins two or more conditions, and display a row only if that row's data satisfies ALL the specified conditions.
  - The **OR** operator joins two or more conditions, and display a row only if that row's data satisfies any of the specified conditions.
  - The **NOT** is a unary operator, and is used to negates a condition.



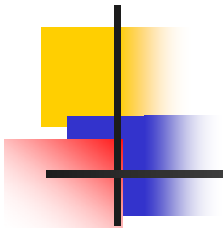
- Arithmetic Operators in SQL

- Another feature of SQL allows the use of arithmetic in queries.
  - The standard arithmetic operators (+, -, /, *) can be applied to numeric values or attributes with numeric domain.
  - The arithmetic operators can be used in expressions in the SELECT and the WHERE clauses to compute numeric values.
  - All attributes that can be computed using arithmetic expressions (such as age from birth date, annual salary from monthly salary) must be eliminated as part of a good design practice in databases.



- SQL Schema and Catalog

- 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, constraints, translations, and character sets. All have same owner.



- SQL Data Definition Statements (DDL)

- CREATE SCHEMA and DROP SCEHMA +
- CREATE TABLE +
- ALTER TABLE +
- DROP TABLE +

-- CREATE SCHEMA and DROP SCHEMA

CREATE SCHEMA [name| AUTHORIZATION creator_id];

Example: CREATE USER COMPANY IDENTIFIED BY password;

DROP SCHEMA name [RESTRICT | CASCADE];

Example: DROP USER COMPANY 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

```
CREATE TABLE table_name
(col_name data_type [NULL | NOT NULL] [,...]);
```

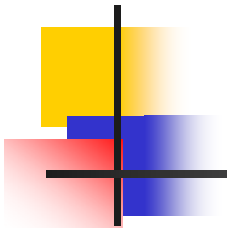
- Creates a table with one or more columns of the specified *data_type*.
- NULL (default) indicates whether column can contain *nulls*.
- With NOT NULL, system rejects any attempt to insert a null in the column.
- Primary keys should always be specified as NOT NULL.
- Foreign keys are often (but not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

--- CREATE TABLE – Example 1

```
CREATE TABLE Employee
      (
          fname            VARCHAR2(15)          NOT NULL
         ,minit            CHAR
         ,lname            VARCHAR2(15)          NOT NULL
         ,ssn              CHAR(9)
         ,bdate            DATE
         ,address          VARCHAR2(50)
         ,sex              CHAR
         ,salary           NUMBER(10,2)          NOT NULL
         ,Superssn         CHAR(9)
         ,dno              NUMBER(3)             NOT NULL
          ,CONSTRAINT employee_ssn_pk PRIMARY KEY(ssn)
          ,CONSTRAINT employee_superssn_fk
             FOREIGN KEY(Superssn)    REFERENCES employee(ssn)
          ,CONSTRAINT employee_dno_fk
             FOREIGN KEY(dno) REFERENCES department(dnumber)
     );
```

--- CREATE TABLE – Example 2

```
CREATE TABLE department
      (
           dname              VARCHAR2(15)       NOT NULL
          ,dnumber            NUMBER(3)          NOT NULL
          ,mgrssn             CHAR(9)
          ,mgrStartDate       DATE
        ,CONSTRAINT department_dnumber_pk
            PRIMARY KEY(dnumber)
        ,CONSTRAINT department_mgrssn_fk
            FOREIGN KEY(mgrssn) REFERENCES employee(ssn)
      );
```

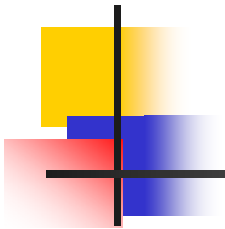


-- DROP TABLE

DROP TABLE tbl_name [RESTRICT | CASCADE]

e.g. DROP TABLE employee;

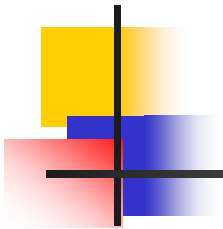
- Removes named table and all rows within it.
- With RESTRICT, if any other objects depend for their existence on continued existence of this table, SQL does not allow request.
- With CASCADE, SQL drops all dependent objects (and objects dependent on these objects).



-- ALTER TABLE

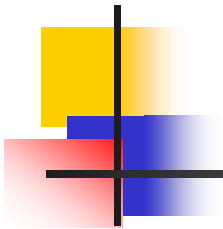
- The ALTER command is a schema modification command.
- It is used to add or drop a column, change a column definition, add or drop table constraints.
- Example:

```
ALTER TABLE COMPANY.EMPLOYEE
MODIFY(lname VARCHAR2(30));
```



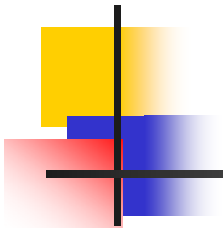
- SQL Data Manipulation Statements (DML)

- INSERT Statement +
- UPDATE Statement +
- DELETE Statement +



-- INSERT Statement

- Definition of INSERT Statement +
- Types of INSERT Statement +
- INSERT and Integrity Constraints +



-- Definition of INSERT Statement

- INSERT is used to add a single row to a table where we specify the relation name and a list of values for the row.

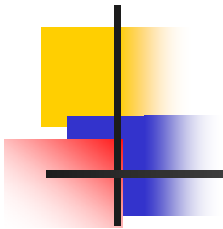
- There are three types of INSERT Statement:
  - INSERT With Column list +
  - INSERT Without Column list +
  - INSERT with SELECT Statement +

--- INSERT with Column list

INSERT INTO table_name (column_list) VALUES (data_value_list);

- Example: **INSERT INTO employee(fname, lname, ssn, salary, dno)**
  VALUES ('Majid', 'Al-Ghamdi', '1111111', 4000, 123);

- *data_value_list* must match *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 *data_value_list* must be compatible with data type of corresponding column.
  - If one of the table columns is omitted from the *column_list* It must also be omitted from the *data_value_list* and make sure it is nullable.

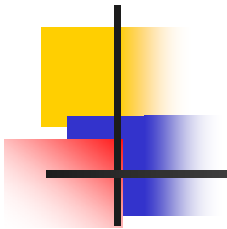


--- INSERT without Colum List

INSERT INTO table_name VALUES (data_value_list);

- **Example:** **INSERT INTO employee**
  VALUES ('Adel', NULL, 'Al-Eid', '222222',
  NULL, NULL, NULL, NULL, NULL, 1);

- *data_value_list* must match the columns of the table as follows:
  - Number of items in the list must be equal to the number of columns of the table.
  - Data type of corresponding items must be compatible.



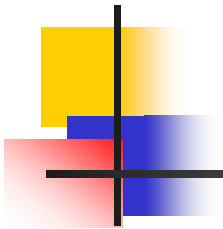
--- INSERT ... SELECT

- Second form of INSERT allows multiple rows to be copied from one or more tables to another:

```
INSERT INTO table_name [ (column_list) ]
SELECT ...
```

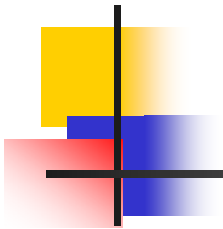
Example:

```
INSERT INTO Table1 (A1, A2, A3)
SELECT B1, B2, B3 FROM Table2;
```



--- INSERT and Integrity Constraints

- A DBMS that fully implement SQL2 should support and enforce all the integrity constraints that can be specified in the DDL.
- A DBMS enforcing NOT NULL will reject an INSERT command in which an attribute declared to be NOT NULL does not have a value.
- A DBMS not supporting referential integrity will allow insertion even if the referential integrity constraint is violated.



-- UPDATE

- Definition +
- Examples
  - Update All Rows +
  - Update Specific Rows +
  - Update Multiple Columns +

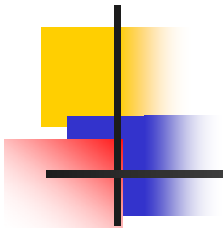
--- UPDATE Definition ...

- The UPDATE command is used to modify attribute values of one or more selected rows.

```
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.

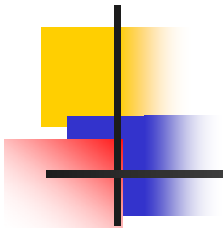
- SET clause specifies names of one or more columns that are to be updated.



... --- UPDATE Definition

- WHERE clause is optional:
  - If omitted, named columns are updated for all rows in table.
  - If specified, only those rows that satisfy *search_condition* are updated.

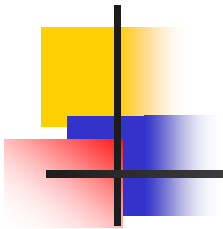
- New *data_value(s)* must be compatible with data type for corresponding column.



---- Example: UPDATE All Rows

Give all employees a 3% pay increase.

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

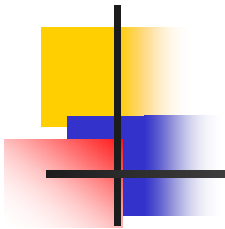


---- Example: UPDATE Specific Rows

- Give all Employees in Department one a 5% pay increase.

```
UPDATE employee
SET salary = salary*1.05
WHERE dno = 1;
```

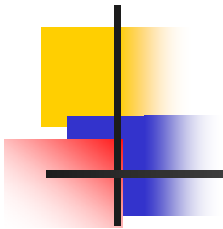
- WHERE clause finds rows that contain data for **dno = 1**. Update is applied only to these particular rows.



---- Example: UPDATE Multiple Columns

- Change Adel's department to 2 and his Salary to 4,000. Assume Adel's ssn = 111;

```
UPDATE employee
SET dno = 2
    , salary = 4000
WHERE ssn = '111';
```



-- DELETE

- DELETE Definition +
- DELETE Example +

--- DELETE Definition

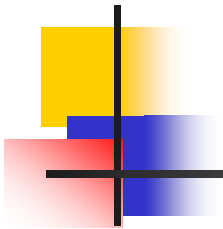
- A DELETE command removes rows from a table and may include a where-clause.

- Rows are explicitly deleted from only one table at a time. However, the deletion may propagate to rows in other tables if referential triggered actions are specified in the referential integrity constraints of the DDL.

DELETE FROM table_name [WHERE search_condition]

- **table_name** can be name of a base table or an updatable view.

- The WHERE clause is optional; if omitted, all rows are deleted from table. But if it is included only those rows that satisfy the **search_condition** are deleted.



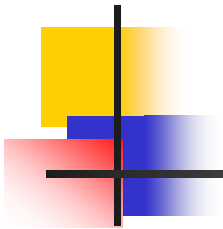
--- Example: DELETE

- Delete all records from employee.

```
DELETE FROM employee;
```

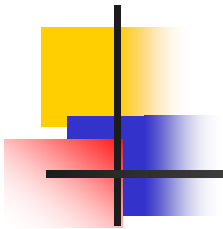
- Delete all employees in department 1.

```
DELETE FROM employee
WHERE dno = 1;
```



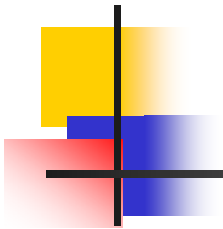
-- SELECT

- SELECT Definition +
- Selecting Columns +
- Selecting Rows +
- Sorting +
- Aggregation +
- Grouping +
- Restricting Groups +
- Aliasing Table Names +
- Nested Queries +
- Join +
- Set Operations +



--- SELECT Definition ...

- SQL has only one statement for retrieving information from a database called the SELECT statement.
- SQL SELECT statement is different from that of Relational Algebra.
- An important distinction between SQL and formal relational model is that SQL allows duplicate rows. Hence an SQL table is not a set but a multiset (some times called a bag) of tuples.

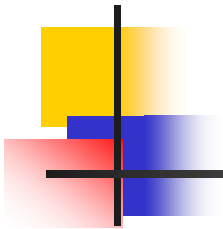


-- SELECT Definition ...

- A SELECT statement can consist up to six clauses.

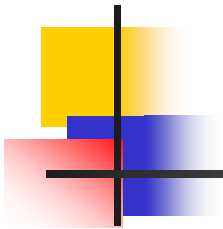
```
SELECT      [DISTINCT | ALL]
            {* | [column_expression [AS new_name]] [,...] }
FROM        table_name [alias] [, ...]
[WHERE      condition]
[GROUP BY   column_list]
[HAVING     condition]
[ORDER By   column_list]
```

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



-- SELECT Definition

- **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.
- **SELECT** Specifies which columns are to appear in output.
- **ORDER BY** Specifies the order of the output.



--- Selecting Columns

- Selecting all columns +
- Selecting Specific columns +
- Selecting Computed columns +
- Renaming Columns +

---- Selecting All Columns

Example 1:

```
SELECT city, year, cars_sold
FROM car_sales;
```

- Can use * as an abbreviation for 'all columns':

Example 2:

```
SELECT *
FROM car_sales;
```

City	Year	Cars_sold
Dhahran	2001	525
Dhahran	2002	456
Riyadh	2001	700
Riyadh	2002	654
Jeddah	2001	921
Jeddah	2002	752
Khobar	2002	

---- Selecting Specific Columns

- Selected columns can be listed as shown in the following example. Notice that the ***year*** column was not selected so it doesn't appear in the output.

Example:

```
SELECT city, cars_sold
FROM car_sales;
```

City	Cars_sold
Dhahran	525
Dhahran	456
Riyadh	700
Riyadh	654
Jeddah	921
Jeddah	752
Khobar	

---- Selecting Computed Columns

- If the value of a car is 100,000 then the total sales per year for each city is computed as follows.

Example:

```
SELECT
  city
 ,year
 ,cars_sold
 ,cars_sold * 100000
FROM car_sales;
```

City	Year	Cars_Sold	*Cars_Sold *100000*
Dhahran	2001	525	***52500000***
Dhahran	2002	456	***45600000***
Riyadh	2001	700	***70000000***
Riyadh	2002	654	***65400000***
Jeddah	2001	921	***92100000***
Jeddah	2002	752	***75200000***
Khobar	2002	0	***0***

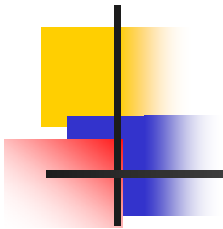
---- Renaming Columns

- The name of the computed column in the last slide cab be changed from cars_sold*100000 to sales as follows.

Example:

```
SELECT
  city
 ,year
 ,cars_sold As Sold
 ,cars_sold * 100000
    AS sales
FROM car_sales;
```

City	Year	Sold	*sales*
Dhahran	2001	525	***52500000***
Dhahran	2002	456	***45600000***
Riyadh	2001	700	***70000000***
Riyadh	2002	654	***65400000***
Jeddah	2001	921	***92100000***
Jeddah	2002	752	***75200000***
Khobar	2002	0	***0***



--- Selecting Rows

- Selecting All Rows +
- Partial match Search +
- Range Search +
- Set Membership Search +
- Pattern matching Search +
- Null Search +
- Removing Duplicate Rows +

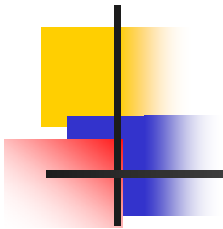
---- Selecting All Rows

- A SELECT statement **without a WHERE clause** selects all rows.

Example:

```
SELECT *
FROM car_sales;
```

City	Year	Cars_Sold
Dhahran	2001	525
Dhahran	2002	456
Riyadh	2001	700
Riyadh	2002	654
Jeddah	2001	921
Jeddah	2002	752
Khobar	2002	



-- Selecting Rows

- To Select certain rows of a table you need to use the WHERE clause of the SELECT statement.
- The WHERE clause has a condition which is a logical expression.
- The Where condition consists of:
  - Comparison Operators
  - Logical Operators
  - Arithmetic Operators
  - Other SQL constructs which will be discussed lated.
- A record to be selected it must make the WHERE logical expression true. In other words it must satisfy the where condition.

---- Partial match Search

- Selecting all the records whose column values match the column values specified in the WHERE clause.

Example1:

```
SELECT  *
FROM    car_sales
WHERE   city = 'Dhahran';
```

City	Year	Cars_Sold
Dhahran	2001	525
Dhahran	2002	456

Example2:

```
SELECT  *
 FROM    car_sales
 WHERE   city = 'Dhahran'
 AND     year > 2001;
```

City	Year	Cars_Sold
Dhahran	2002	456

---- Range Search

Selecting all the records whose column values is between the values specified in the WHERE cluause.

Example:

```
SELECT *
FROM car_sales
WHERE cars_sold >= 525
AND    cars_sold <= 752;
```

OR

```
SELECT *
FROM car_sales
WHERE cars_sold
BETWEEN 525 AND 752;
```

City	Year	Sold
Dhahran	2001	525
Riyadh	2001	700
Riyadh	2002	654
Jeddah	2002	752

- BETWEEN test includes the endpoints of range. NOT BETWEEN list the one not in the range.

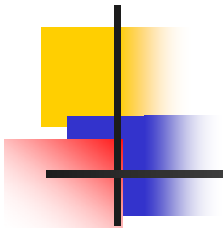
---- Set Membership Search ...

Selecting all the records whose column value is a member of the set specified in the WHERE clause.

Example:

```
SELECT *
FROM car_sales
WHERE city
IN
('Dhahran', 'Riyadh');
```

City	Year	Sold
Dhahran	2001	525
Dhahran	2002	456
Riyadh	2001	700
Riyadh	2002	654



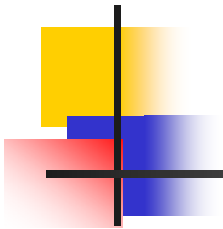
... ---- Set Membership Search

Selecting all the records whose column value not a member of the set specified in the WHERE clause.

Example:

```
SELECT *
FROM car_sales
WHERE city
NOT IN
('Dhahran', 'Riyadh');
```

City	Year	Sold
Jeddah	2001	921
Jeddah	2002	752
Khobar	2002	



---- Pattern Matching Search ...

- SQL has two special pattern matching symbols:
  - **%** : sequence of zero or more characters;
  - _ (underscore): any single character.

- **LIKE** '%dd%' means a sequence of characters of any length containing '*dd*'.

---- Pattern matching Search

Selecting all the records whose column value match the pattern specified in the WHERE clause.

Example:

```
SELECT *
FROM car_sales
WHERE
city LIKE  'J%'
```

City	Year	Sold
Jeddah	2001	921
Jeddah	2002	752

Example:

```
SELECT *
FROM car_sales
WHERE
city LIKE  '%dd%'
```

City	Year	Sold
Jeddah	2001	921
Jeddah	2002	752

---- NULL Search

Example 1: Select all cities where the number of cars sold is **unkown.**

```
SELECT city
FROM car_sales
WHERE cars_sold IS NULL;
```

City
Khobar

Example 2: Select all cities where the number of cars sold **is kown.**

```
SELECT city
FROM car_sales
WHERE cars_sold IS NOT NULL;
```

City
Dhahran
Dhahran
Riyadh
Riyadh
Jeddah
Jeddah

---- Removing Duplicate Rows

Example1:

```
SELECT city
FROM car_sales
```

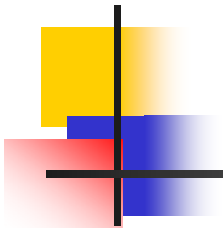
City
Dhahran
Dhahran
Riyadh
Riyadh
Jeddah
Jeddah
Khobar

Example2:

```
SELECT DISTINCT city
FROM car_sales
```

City
Dhahran
Riyadh
Jeddah
Khobar

Using **DISTINCT** in the SELECT clause removes duplicate rows from the output table



---- Sorting

- The ORDER BY clause specifies an order for displaying the result of a query.
  - SQL allows the user to order the tuples in the result of a query by the values of one or more attributes; the default order is ascending or increasing.
  - The keyword **DECS** is specified to sort in a descending order of values while the keyword **ASC** can be used to specify ascending order explicitly.
  - The sorting will be applied alphabetically or numerically depending on the type of the column attribute.

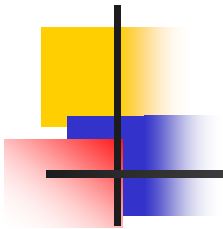
---- Example: Sorting

Example:

The following SELECT statement sorts the car_sales table in ascending order of city and descending order of car_sales columns

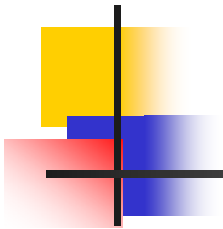
```
SELECT *
FROM car_sales
ORDER BY city asc, car_sales desc;
```

City	Year	Cars_Sold
Dhahran	2001	525
Dhahran	2002	456
Jeddah	2001	921
Jeddah	2002	752
Khobar	2002	
Riyadh	2001	700
Riyadh	2002	654



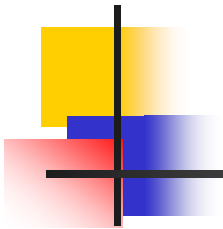
--- Aggregation ...

- 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.



... --- Aggregation ...

- 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.



... --- Aggregation

- 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 with an aggregate function. For example, following is illegal:

```
SELECT city, COUNT(*)
 FROM car_sales;
```

---- Example : COUNT

- How many rows are there in the car_sales table?

```
SELECT COUNT(*) as Rows
FROM car_sales
```

Rows
7

- How many cities are there in the car_sales table?

```
SELECT
COUNT(DISTINCT city)
as city
FROM car_sales
```

city
4

---- Example : SUM

- Find the total number of all the cars sold from the car_sales table?

```
SELECT
SUM(cars_sold) as cars_sold
FROM car_sales
```

Cars_sold
4008

- Find the number of all the cars_sold in Dhahran from the car_sales table?

```
SELECT
SUM(cars_sold) as Dah_cars
FROM car_sales
WHERE city = 'Dhahran'
```

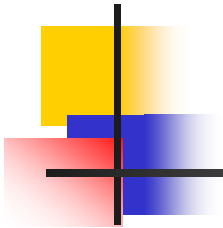
Dah_cars
981

---- Example: MIN, MAX, AVG

- Find the minimum, maximum, and average cars_sold per year and per city form the car_sales table

```
SELECT MIN(cars_sold)  as Min_sold
       , MAX(cars_sold) as Max_sold
       , AVG(cars_sold)  as Avg_sold
FROM car_sales
WHERE car_sales IS NOT NULL;
```

Min_sold	Max_sold	Avg_sold
456	921	668



--- Grouping

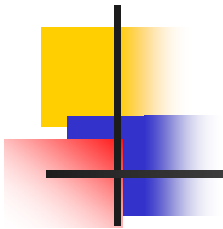
- Use **GROUP BY** clause to get sub-totals.
- SELECT and GROUP BY closely integrated: each item in SELECT list must be **single-valued per group**, and SELECT clause may only contain:
  - Column names.
  - Aggregate functions.
  - Constants.
  - An expression involving combinations of the above.
- All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

---- Example: Grouping

- Find the total cars sold in each city from the car_sales table.

```
SELECT city, SUM(cars_sold) as cars
FROM car_sales
WHERE cars_sold IS NOT NULL
GROUP BY city
ORDER BY SUM(cars_sold) ;
```

City	Cars
Dhahran	981
Riyadh	1354
Jeddah	1637



--- Restricting Groups

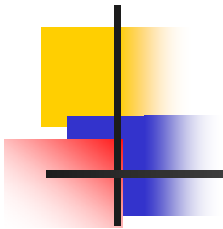
- **HAVING clause** is designed for use with GROUP BY clause to restrict groups that appear in final result table.
- Similar to WHERE, but WHERE filters individual rows whereas HAVING filters groups.
- Column names in HAVING clause must also appear in the GROUP BY list or be contained within an aggregate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

---- Example: Restricting Groups

- Find the cities who sold a total of more than 1000 cars from the car_sales table.

```
SELECT city, SUM(cars_sold) as cars
FROM car_sales
WHERE cars_sold IS NOT NULL
GROUP BY city
HAVING SUM(cars_sold) > 1000 ;
```

City	Cars
Riyadh	1354
Jeddah	1637



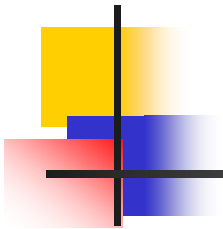
-- Aliasing Table Names

- A table alias is created by directly placing an alias after the table name in the FROM clause.
- The advantage of using a table alias when performing JOIN is readily apparent when we discuss JOIN later.

- For example in the following example we will refer to departments table as **d or dept**.

```
SELECT d.dname
FROM   departments d
WHERE d.dno = 1;
```

```
SELECT dept.dname
FROM   departments dept
WHERE dept.dno = 1;
```

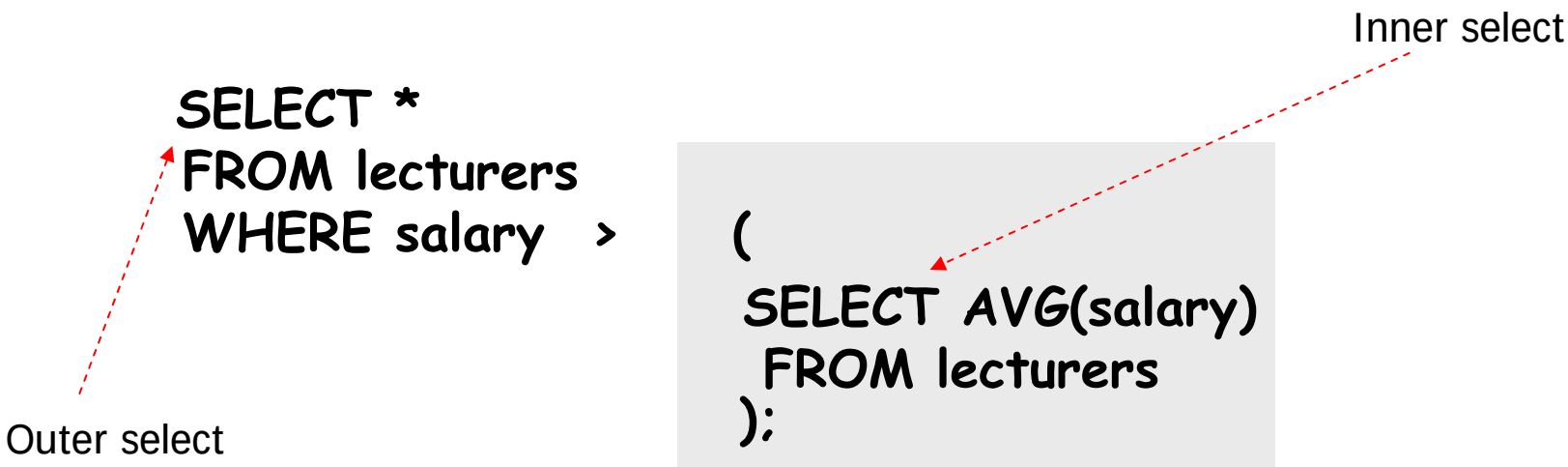


--- Nested queries

- 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 ***nested query*** or a ***subquery***.
- Subselects may also appear in INSERT, UPDATE, and DELETEs.

---- Example: Nested queries

- From the Lecturer table, select lecturers whose salary is above average.
- Cannot write 'WHERE salary > avg(salary)'.

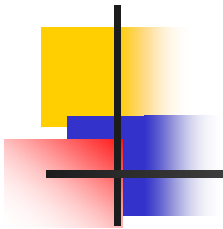


- The Inner select is done before the outer select.

---- Nested query: Example

- List the names of all Lecturers who are in the ICS department

```
SELECT lname
FROM lecturers
WHERE dno  IN (
                SELECT dno
                FROM department
                WHERE dname  = 'ICS'
              );
```



---- Nested Query 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.

---- Nested Query: Example

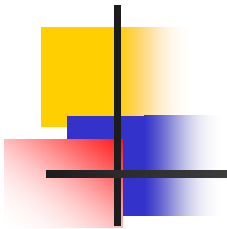
- Find lecturers whose salary higher than the salary of at least 1 COE lecturer.

```
SELECT *
FROM Lecturers
WHERE salary >  (
                    SELECT min(salary)
                    FROM lecturers
                    WHERE dno =    (
                                        SELECT DNO
                                        FROM department
                                        WHERE dname = 'COE'
                                    )
                );
```

---- Nested Query: Example

- Find lecturers whose salary higher than the salary of every COE lecturer.

```
SELECT *
FROM Lecturers
WHERE salary >  (
        SELECT max(salary)
        FROM lecturers
        WHERE dno =    (
                SELECT DNO
                FROM department
                WHERE dname = 'COE'
                )
    );
```



-- Join

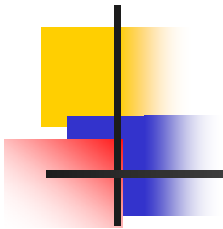
- Can use subqueries provided result columns come from same table.
- If result columns come from more than one table must use a join.
- To perform join, include more than one table in FROM clause.
- Use comma as separator and typically include WHERE clause to specify join column(s).
- Also possible to 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: Join (Inner Join) ...

- The default type of join is inner join, where arow is included in the result only if matching row exists in the other relation.
- List each lecturer's name and his department name.

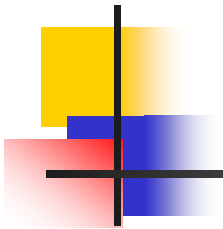
```
SELECT a.lname, b.dname
FROM   lecturers a,
       departments b
WHERE a.dno = b.dno;
```

Lname	dname
Ahmed	ICS
Amin	COE
Hani	ICS
Ageel	ICS
Yousef	COE
Khalid	COE



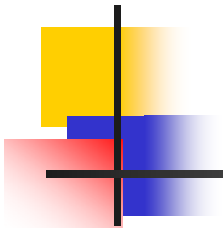
... Example: Join (Inner Join)

- To obtain correct rows, include only those rows from both tables that have identical values in the dno columns: a.dno = b.dno.
- These two columns are the matching columns for two tables.
- This type of join is also called **inner join** and they equivalent to equi-join in relational algebra.



---- Computing a Join

- Procedure for generating results of a SELECT with a 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 ...

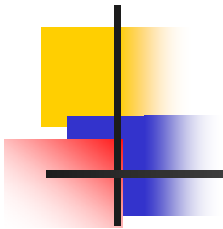
- With an inner join, if one row of a table is unmatched, row is omitted from result table.
- The outer join operations retain rows that do not satisfy the join condition.
- There are three types of OUTER JOIN
  - Left Outer Join
  - Right Outer Join
  - Full Outer Join
- Lets discuss inner join then we will come back to outer join.

... ---- Outer Join ...

- Inner join of departments and lecturers tables will result in the following output.

```
SELECT a.*, b.*
FROM lecturers a,
     Departments b
WHERE a.dno = b.dno
```

Lid	Lname	dno	salary	dno	dname
1	Ahmed	1	4000	1	ICS
2	Amin	2	3700	2	COE
3	Hani	1	4200	1	ICS
4	Ageel	1	4000	1	ICS
5	Yousef	2	3500	2	COE
6	Khalid	2	4500	2	COE



... ---- Outer Join ...

- Result table has two rows where the dno are the same.
- There are no rows corresponding to NW or Abdella.
- To include unmatched rows in result table, use an outer join.

Input Tables

---- Example: Left Outer Join

- If We want to Include in the output table the lecturers whose department is unknow we rewrite our previous query as follows

```
SELECT a.*, b.*
FROM lecturers a,
     Departments b
WHERE b.dno(+) = a.dno
```

Lid	Lname	dno	salary	dno	dname
1	Ahmed	1	4000	1	SE
2	Amin	2	3700	2	SWE
3	Hani	1	4200	1	ICS
5	Ageel	1	4000	1	SWE
6	Yousef	2	3500	2	COE
7	Khalid	2	4500	2	COE
4	**Addella**		**4300**		

---- Example: Right Outer Join

- If We want to Include in the output table the departments with no lecturers we rewrite our previous query as follows

```
SELECT a.*, b.*
FROM lecturers a,
     Departments b
WHERE a.dno = b.dno(+)
```

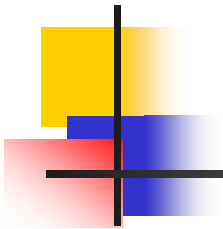
Lid	Lname	dno	salary	dno	dname
1	Ahmed	4	4000	4	SE
2	Amin	3	3700	3	SWE
3	Hani	1	4200	1	ICS
5	Ageel	3	4000	3	SWE
6	Yousef	2	3500	2	COE
7	Khalid	2	4500	2	COE
				5	**NW**

---- Example: Full Outer Join

- If We want to Include in the output table the departments with no lecturers and the lecturers with unknow departments we rewrite our previous query as follows

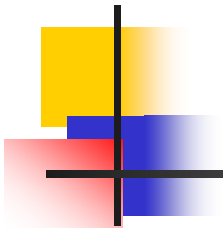
```
SELECT a.*, b.*
FROM lecturers a,
     Departments b
WHERE a.dno = b.dno(+)
UNION
SELECT a.*, b.*
FROM lecturers a,
     Departments b
WHERE a.dno(+) = b.dno;
```

Lid	Lname	dno	salary	dno	dname
1	Ahmed	4	4000	4	SE
2	Amin	3	3700	3	SWE
3	Hani	1	4200	1	ICS
5	Ageel	3	4000	3	SWE
6	Yousef	2	3500	2	COE
7	Khalid	2	4500	2	COE
				5	**NW**
4	**Abdella**		**4300**		



---- Characteristic of Outer Join

- Left Outer Join:
  - Includes those rows of first (left) table unmatched with rows from second (right) table.
  - Columns from second table are filled with NULLs.
- Right outer Join :
  - includes those rows of second (right) table that are unmatched with rows from first (left) table.
  - Columns from first table are filled with NULLs.
- Full Outer Join:
  - Is the UNION of both left and right outer joins.



-- Union, Intersect, and Difference

- 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*.
- If ALL specified, result can include duplicate rows

---- Example: Use of UNION ...

- **List all the ICS and COE faculty salaries. Remove duplicates**

```
SELECT salary
FROM lecturers
WHERE dno = 1
UNION
SELECT salary
FROM lecturers
WHERE dno = 2;
```

- **List all the ICS and COE faculty salaries. Include duplicates**

```
SELECT salary
FROM lecturers
WHERE dno = 1
UNION ALL
SELECT salary
FROM lecturers
WHERE dno = 2;
```

... ---- Example: Use of UNION

- **List all the ICS and COE faculty salaries. Remove duplicates**

```
SELECT salary
FROM lecturers
WHERE dno =
( SELECT dno
  FROM departments
  WHERE dname= 'ICS'
)
UNION
SELECT salary
FROM lecturers
WHERE dno =
( SELECT dno
  FROM departments
  WHERE dname= 'COE'
)
```

- **List all the ICS and COE faculty salaries. Include duplicates**

```
SELECT salary
FROM lecturers
WHERE dno =
( SELECT dno
  FROM departments
  WHERE dname= 'ICS'
)
UNION ALL
SELECT salary
FROM lecturers
WHERE dno =
( SELECT dno
  FROM departments
  WHERE dname= 'COE'
)
```

... ---- Example: Use of DIFFERENCE

- **List salaries that are taken by ICS and not COE lecturers.**

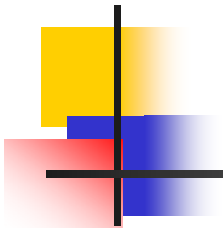
```
SELECT salary
FROM lecturers
WHERE dno = (
                    SELECT dno
                    FROM departments
                    where dname= 'ICS'
                )
MINUS
SELECT salary
FROM lecturers
WHERE dno = (
                    SELECT dno
                    FROM departments
                    WHERE dname= 'COE'
                )
```

... ---- Example: Use of INTESECTION

- **List salaries that are taken by both COE and ICS lecturers.**

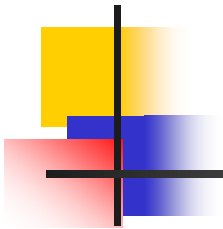
```
SELECT salary
FROM lecturers
WHERE dno = (
                SELECT dno
                FROM departments
                where dname= 'ICS'
            )
INTERSECT
SELECT salary
FROM lecturers
WHERE dno = (
                SELECT dno
                FROM departments
                WHERE dname= 'COE'
            )
```

Produces result tables from both queries and creates single result table consisting of those rows that are common to both result tables.



- Other SQL Operators

- IN (covered)
- BETWEEN (covered)
- LIKE (covered)
- ANY (SOME) +
- ALL +
- EXISTS +
- NOT EXISTS +



-- ANY (SOME) 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 using the SOME Operator

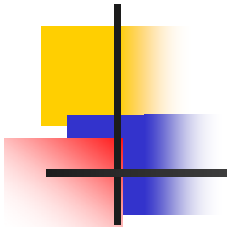
- Find lecturers whose salary higher than the salary of at least 1 COE lecturer.

```
SELECT *
FROM Lecturers
WHERE salary > SOME (
        SELECT salary
        FROM lecturers
        WHERE dno =    (
                SELECT DNO
                FROM department
                WHERE dname = 'COE'
                )
    );
```

--- Example Using the ALL Operator

- Find lecturers whose salary higher than the salary of every COE lecturer.

```
SELECT *
FROM Lecturers
WHERE salary > ALL   (
                        SELECT salary
                        FROM lecturers
                        WHERE dno =    (
                                           SELECT DNO
                                           FROM department
                                           WHERE dname = 'COE'
                                       )
                    );
```



-- EXISTS and NOT EXISTS

- EXISTS and NOT EXISTS are for use only with subqueries specially with **correlated subqueries**. **A correlated subquery** is a subquery where some attributes of the outer select are used in the inner select.
- They produce a simple true/false result.
- EXISTS is true if and only if there exists at least one row in result table returned by subquery.
- It is false if subquery returns an empty result table.
- NOT EXISTS is the opposite of EXISTS.
- 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.

--- Example using the EXISTS Operator

- Find all ICS lecturers.

```
SELECT *
FROM lecturers a
WHERE EXISTS
                    (
                        SELECT  1
                    FROM   department  b
                    WHERE a.dno = b.dno
                        AND    b.dname = 'ICS'
                    );
```

--- Example using the NOT EXISTS Operator

- Find all non ICS lecturers.

```
SELECT *
FROM lecturers a
WHERE NOT EXISTS
                (
                    SELECT  1
                  FROM   department  b
                  WHERE a.dno = b.dno
                      AND    b.dname = 'ICS'
                );
```