Bold black for commands/keywords

Blue: *explanations*

Red: *data/table/column names*

🟧 SQL Course Notes – Updated VER

🔹 DDL – Data Definition Language

Commands:
CREATE, ALTER, DROP, RENAME, TRUNCATE

🔸 CREATE Statement

create table staff (

civilid **number**(3),

staffname **varchar**(30),

email **varchar**(40));

describe staff; → show the structure of the table.

🔸 ALTER Statement

Add column:

alter table student

add gender **varchar**(6); --> to add new column.

-To add a primary key:

alter table student

add primary key(id);

-Alter Column DataType:

```
ALTER TABLE table_name
MODIFY COLUMN column_name datatype;
```

EX1: **Alter** teacher Teacher_name column **Drop** DOB;

EX2: **Alter** Table **Teacher Modify** Teacher_name **char**;

-Drop column:

ALTER TABLE teacher

DROP COLUMN DOB;

-Modify column type:

ALTER TABLE teacher

MODIFY teacher_name CHAR;

🔹 **DML – Data Manipulation Language**

Commands:
INSERT, **DELETE**, **UPDATE**, **SELECT**

🔸 **INSERT Statement**

INSERT INTO student (id, name, location, mob, gender)

VALUES (200, 'Ahmed', 'Muscat', 90989876, 'Male');

Use single quotes for string (varchar) values.

🔸 **SELECT Statement**

Selects columns or all data from a table.

select columnName from Tablename; --> to show the data

select Id from student;

select id,gender from student;

select * from student; --> the * means all

String concatenation:

select empid || empname from employee; --> view data of 2 columns together

select * from emplyee **where** empid=10; --> will get one record

select * from emplyee **where** empname='ahmed'; --> single quotes for strings.

select * from emplyee **where** empid > 10; --> all records greater than 10

select * from emplyee **where** empid >=10; -->10 and more

select * from teacher **where** dob = '01-jan-2020'

With alias:

select teacher_name as staff from teacher

🔹 **Constraints**

Types:

- NOT NULL
- UNIQUE
- PRIMARY KEY
- FOREIGN KEY
- CHECK

🔸 **Column-Level Constraint Example**

```
CREATE TABLE empTLC (
 Empno NUMBER(4) CONSTRAINT emp_empno_pk PRIMARY KEY,
 Ename VARCHAR(20) CONSTRAINT emp_ename_nnTL NOT NULL,
 birthday DATE CONSTRAINT emp_birthday_nnTL NOT NULL,
 Salary NUMBER(6) CONSTRAINT emp_salary_nnTL NOT NULL,
 Email VARCHAR(25),
 CONSTRAINT emp_email_ukTL UNIQUE
);
```

- **Add a CHECK Constraint**

ALTER TABLE emp1

MODIFY salary NUMBER(6) **CONSTRAINT** emp_salary_n33

CHECK(salary > 2500);

- **SELECT Constraints**

SELECT * FROM user_constraints;

- **Foreign Key – Column Level**

create table Emp(

Empno Number(4) constraint emp_empno_pk Primary Key,

Ename Varchar(15) constraint emp_ename_nn Not Null,

BirthDate Date constraint emp_bdate_nn Not Null,

Salary Number(6) constraint emp_salary_nn Not Null,

Deptno Number(4) References Dept(deptno)

); # we add the FK constrain whitin the column it self.

- **Foreign Key – Table Level**

CREATE TABLE Emp (

Deptno NUMBER(4),

CONSTRAINT emp_deptno_fk FOREIGN KEY(deptno) REFERENCES Dept(deptno)

); # we added the constraint at the end.

- **Multiple CHECK Constraints**

CREATE TABLE personSal (

Person_ID INT,

Last_Name VARCHAR(255),

FirstName VARCHAR(255),

salary NUMBER(8,2),

CONSTRAINT Personal_min_sal_ck CHECK (salary > 0),

CONSTRAINT Personal_max_sal_ck CHECK (salary < 1000)

);

* we can add multiple constrain to the same columns, each with a unique constraint name.

🔹 **Managing Constraints**

- **Add constraint:**

Alter table personSal

Add constraint personsal_pid_pk primary key (person_ID);

- **Drop constraint:**

ALTER TABLE personSal

DROP CONSTRAINT personsal_max_sal_ck;

- **Disable constraint:**

ALTER TABLE personsal

DISABLE CONSTRAINT **personsal_pid_pk**;

- **Enable:**

ALTER TABLE employees

ENABLE CONSTRAINT emp_id;

- **Modify NOT NULL:**

ALTER TABLE stud

MODIFY (email CONSTRAINT stud_email_nn NOT NULL);

🔹 Insert, Update, Delete

- **Insert record:**

```
INSERT INTO personsal1 (person_id, last_name, firstname, salary)
VALUES (222, 'Ahmed', 'Mohammed', 2000);
```

- **Insert without column list (ensure order matches):**

```
INSERT INTO personsal1
VALUES (333, 'Ali', 'Mohammed', 2000);
```

** order of the values should match columns order in our table.

- **Copy table:**

```
CREATE TABLE copy_person AS
(SELECT * FROM personsal1);
```

- **Update existing data/record:**

```
UPDATE personsal1
SET Salary = 2000
where person_id= 111;
```

- **Delete value:**

```
DELETE FROM personsal1
WHERE person_id = 111;
```

Deleting multiple values with a condition:

```
DELETE FROM personsal1
WHERE person_id > 250;
```

🔹 **Boolean Values in SQL (Logic Check Notes)**

Can be represented as:

- TRUE / FALSE
- ON / OFF
- 1 / 0
- YES / NO

🟧 SQL Logical Conditions & Filtering (Where Clause + Operators)

🔹 **DEFAULT Values**

DEFAULT SYSDATE

- Used to assign the current system date to a column, usually in a DATE column.
- Can be applied when creating or later altering a table.
- If value is not specified during insert, the default will be used.

◇ WHERE Clause – Filtering Rows

Used with:

- **Comparison operators**
- **Logical operators**
- **Pattern matching**
- **NULL checks**

◇ Comparison Operators

- **Not Equal**

SELECT empid, empname, deptid

FROM employee

WHERE deptid **!=** 90;

- **Equals**

```
SELECT first_name, last_name
FROM employees
WHERE last_name = 'Taylor';   ** Case-sensitive search for string values.
```

- **Greater Than or Equal**

```
SELECT first_name, last_name
FROM my_employee
WHERE hire_date >= '01-Jan-2000';
```

- **Less Than or Equal**

```
SELECT empid, empname
FROM employee1
WHERE salary <= 3000;
```

- **Between**

```
SELECT empid, empname
FROM employee1
WHERE salary BETWEEN 500 AND 2500;
```

*** Avoid rewriting it as salary >= 9000 AND salary <= 11000.*

◇ **IN Operator**

- **Example:**

```
SELECT city, state_province, country_id
FROM locations
WHERE country_id IN ('UK', 'CA');
```

◇ LIKE Operator (Pattern Matching)

Select last_name from employee

Where last_name like 'A%'; **search for all names start with of A.

Select last_name from employee

Where last_name like '%d'; **search for all names Ended with of D

Select last_name from employee

Where last_name like '%d%'; **search for all names have letter D anywhere in middle.

Select last_name from employee

Where last_name like '_a%'; **search for all names have letter A in the second position

Select last_name from employee

Where last_name like '__a%'; **search for all names have letter A in the 3rd position

Select last_name from employee

Where last_name like '____'; **search for all names have 4 letters

---

◇ NULL Handling

- IS NOT NULL

SELECT last_name, manager_id

FROM employees

WHERE manager_id **IS NOT NULL**;

- **IS NULL**

SELECT last_name, commission_pct

FROM employees

WHERE commission_pct **IS NULL**;

◇ Logical Operators: AND, OR, NOT

◇ OR Operator

SELECT empid, empname

FROM employee1

WHERE empid >= 9000 **OR** salary <= 11000;

** *Returns rows if **either** condition is true.*

Ex2:

SELECT last_name, hire_date, job_id

FROM employees

WHERE hire_date > '01-Jan-1998' **OR** job_id LIKE 'SA%';

◇ AND Operator

SELECT empid, empname

FROM employee1

WHERE empid >= 9000 **AND** salary <= 11000;

** *Both conditions must be true.*

◇ NOT Operator

```
SELECT department_name, location_id
FROM employees
WHERE location_id NOT IN (1700, 1800);
```

```
SELECT *
FROM personsal1
WHERE NOT last_name = 'Ali';
```

◇ Operator Precedence (AND > OR)

Always evaluated as:

** **AND** → before **OR**

```
SELECT *
FROM personsal1
WHERE last_name = 'Ahmed' AND person_id = 222 OR salary = 333;
```

```
SELECT *
FROM personsal1
WHERE last_name = 'Ahmed' OR person_id = 227 AND salary = 333;
```

** *To avoid confusion, use parentheses for clarity.*

🟧 Ordering, Aliasing, and SQL Single Row Functions

🔹 Ordering Data

- **Ascending Order** (default):

```
SELECT last_name, hire_date
FROM employees
ORDER BY last_name;
```

- **Descending Order**:

```
SELECT last_name, hire_date
FROM employees
ORDER BY last_name DESC;
```

With Column Aliases:

```
SELECT last_name AS "Name", hire_date AS "Date Started"
FROM employees
ORDER BY "Name", hire_date;
```

- **Multiple Columns**:

```
SELECT department_id, last_name
FROM employees
WHERE department_id <= 50
ORDER BY department_id, last_name DESC;
```

🟧 CH6: SQL Functions

🔹 DUAL Table

SELECT (2 + 2) FROM **dual**;

*** The DUAL table is a dummy table used for calculations or function testing.*

🔹 Character Functions

- LOWER()

SELECT **LOWER**(last_name) FROM employees;

*** Converts characters to lowercase.*

- UPPER()

SELECT **UPPER**(last_name) FROM employees;

*** Converts characters to uppercase.*

- INITCAP()

SELECT **INITCAP**(last_name) FROM employees;

*** Capitalizes the first letter of each word.*

- CONCAT()

SELECT **CONCAT**(first_name, last_name) FROM employees;

SELECT **CONCAT**('Hello', 'World') FROM dual;

*** Joins two strings.*

With a custom string:

SELECT **CONCAT**(first_name, 'And', last_name) FROM employees;

- **Nested:**

SELECT **LOWER**(**CONCAT**(first_name, last_name)) FROM employees;

🔹 Substring & Length Functions

- **SUBSTR()**

SELECT **SUBSTR**('HelloWorld', 1, 5) FROM dual; -- 'Hello'

SELECT **SUBSTR**('HelloWorld', 6) FROM dual;   -- 'World'

*** Extracts substring starting at position with optional length.*

- **LENGTH()**

SELECT **LENGTH**('HAHAHA') FROM dual;

SELECT **LENGTH**(last_name) FROM employees;

*** Returns number of characters.*

🔹 Position Function

- **INSTR()**

SELECT **INSTR**('HelloWorld', 'W') FROM dual; -- 6

SELECT last_name, **INSTR**(last_name, 'a') FROM employees;

*** Finds the position of a substring.*

Q: **Find the position of letter b in the first name of employees ?**

Select First_name,instr(First_name, 'b') from employees;

🔹 Padding Functions

LPAD(): *Pads the left side of a string.*

SELECT **LPAD**('HelloWorld', 15, '-') FROM dual;

** *result must be total 15 letters so if it's less will add the dashes.*

Output: -----HelloWolrd

SELECT **LPAD**(last_name, 10, '*') FROM employees;

** *Total 10 digits or chars.*

output: ******abel

RPAD(): *Pads the Right side of a char string*

SELECT **RPAD**('Hind', 8, '3') FROM dual;

Output : Hind3333

SELECT RPAD('rahma', 20, 'Oman') FROM dual;

Output: rahmaOmanOmanOman

🔹 Trimming Characters

TRIM() *: Remove all specific chars from beginning , end , or both*

SELECT **TRIM**(LEADING 'a' FROM 'abcba') FROM dual;

-- 'bcba'

SELECT **TRIM**(TRAILING 'a' FROM 'abcba') FROM dual;

-- 'abcb'

SELECT **TRIM**(BOTH 'a' FROM 'abcba') FROM dual;

-- 'bcb'

** *Removes specific characters from start, end, or both sides.*

🔹 Replacing Text

REPLACE(): *Replaces characters in a string.*

SELECT **REPLACE**('Jack and Joe', 'J', 'Bl') FROM dual;

-- 'Black and Bloe'

SELECT **REPLACE**('HOOOWORLD', 'O') FROM dual;

^ will delete O (HWRLD) "Can't replace O with O "

-- 'HWRLD'

🟧 User Input, Rounding, Date Functions in SQL

SELECT first_name, last_name, salary, department_id

FROM employees

WHERE department_id = **:**enter_dept_id;

*** Prompts the user to input a department ID before executing the* query.

SELECT tid, tname, salary

FROM teacherhct

WHERE tid = :enter_teacher_id;

*** User enters the Teacher ID during execution.*

Numeric Functions

- **ROUND**(column, decimal_places)

Example	Output	Note
ROUND(1.95, 0)	2	Round to nearest integer
ROUND(1.94, 1)	1.9	Round to 1 decimal
ROUND(1.94, 2)	1.94	Already precise
ROUND(125, -1)	130	Round to nearest tens
ROUND(123.456, -1)	120	Round left of decimal

- **TRUNC**(column, decimal_places)

Example	Output	Note
TRUNC(45.926, 2)	45.92	Keep 2 decimals
TRUNC(45.926, 1)	45.9	Keep 1 decimal
TRUNC(45.926, 0)	45	Truncate to integer
TRUNC(45.926, -1)	40	Truncate to nearest 10
TRUNC(45.926, -2)	0	Truncate to nearest 100

- MOD(n, d) – Modulus Operator

SELECT **MOD**(6, 2) FROM dual; -- 0

SELECT **MOD**(5, 2) FROM dual; -- 1

** *Returns the remainder after division.*

🟧 System Date and Calculations

SELECT **SYSDATE** FROM dual;

*** Shows current system date.*

SELECT last_name, **hire_date + 10** FROM employees;

*** Adds 10 days to each hire date.*

SELECT last_name, **(SYSDATE - hire_date) / 7** AS weeks

FROM employees;

*** Returns time since hire in weeks.*

SELECT last_name, **(SYSDATE - hire_date) / 365** AS years

FROM employees;

*** Returns time since hire in years.*

🟧 Date Functions

- **MONTHS_BETWEEN(date1, date2)**

*** Number of months between two dates.*

SELECT last_name, hire_date, **MONTHS_BETWEEN**(SYSDATE, hire_date)

FROM employees;

SELECT **MONTHS_BETWEEN**('12-OCT-2021', '12-OCT-2020') FROM dual; -- 12

- **Filter example:**

SELECT last_name, hire_date

FROM employees

WHERE **MONTHS_BETWEEN**(SYSDATE, hire_date) > **24**;

- **ADD_MONTHS(date, number_of_months)**

SELECT **ADD_MONTHS**(SYSDATE, 1) FROM dual;

-- Next month from current date

SELECT **ADD_MONTHS**(SYSDATE, 12) AS next_year FROM dual;

- **NEXT_DAY(date, 'DayName')**

SELECT **NEXT_DAY**(SYSDATE, 'Saturday') AS "Next Saturday" FROM dual;

SELECT **NEXT_DAY**(SYSDATE, 'Monday') AS "Next Monday" FROM dual;

** All week days are Case sensitive.*

- LAST_DAY(date)

SELECT **LAST_DAY**(SYSDATE) AS "Last Day in Month" FROM dual;

SELECT **LAST_DAY**(**ADD_MONTHS**(SYSDATE, 1)) AS "Last Day in Next Month" FROM dual;

Date Rounding

- ROUND(date, 'Month'):

Month=30 days

*** If day <16 → rounds down, else → next month*

SELECT **ROUND**(DATE '2020-02-12', 'Month') FROM dual;

-- 01-FEB-2020

SELECT **ROUND**(DATE '2020-02-17', 'Month') FROM dual;

-- 01-MAR-2020

- ROUND(date, 'Year')

Year=12 months

*** If month <6 → rounds to Jan this year, else → Jan next year*

SELECT **ROUND**(DATE '2020-02-12', 'Year') FROM dual;

-- 01-JAN-2020

SELECT **ROUND**(DATE '2020-08-17', 'Year') FROM dual;

-- 01-JAN-2021

🟧 CH6.5: Date Truncation & Conversion Functions

TRUNC(date, 'format')

- Within the Month

Select hire_date,**trunc**(hire_date,'Month') from employees

Where department_id=50;

** Example Output:*

16-NOV-1999 → 01-NOV-1999

17-OCT-1995 → 01-OCT-1995

- Within the Year

SELECT hire_date, **TRUNC**(hire_date, 'Year')

FROM employees

WHERE department_id = 50;

*** Truncates the date to the first day of the same year.*

** Example Output:*

16-NOV-1999 → 01-JAN-1999

17-OCT-1995 → 01-JAN-1995

🟧 Date → Character Conversion (TO_CHAR)

Format:

TO_CHAR(date_column, 'format_model')

Select **To_Char**(Hire_date,'Mon*th dd, yyyy') from employees;*

^ Output (June 07, 1994)

Common Date Format Models

Format	Output Example	Description
Month	June	Full month name
MON	JUN	3-letter abbreviation
DD	07	Day of month (2 digits)
D	3	Day of week (1–7)
DDD	322	Day of the year
YYYY	2024	Year
HH24	14	Hour (24-hour)
MI	53	Minutes
SS	11	Seconds
AM/PM	AM	Morning / Evening

Format	Output Example	Description
SP	Seven	Spells out numbers
TH	7th	Ordinal form
FM	Removes leading 0s	(e.g., 07 → 7)

◆ Examples

Select **To_char (hire_date , 'month, ddth,yyyy')** from employees;

Select **To_char (sysdate , 'month , ddspth , yyyy')** from dual;

Select **To_char (hire_date , 'MON , ddth , year ')** from employees;

Select **To_char (hire_date , 'mm , ddth , year ')** from employees;

Select **To_char (hire_date , 'fmMMfm ddth , year ')** from employees;

***Note:*

DDD --> Day of the week

DD --> Day of the month

D --> day of the week

- Detailed Date + Week Formatting

SELECT **TO_CHAR(hire_date, 'DY fmMMfm ddth, Year')** FROM employees;

SELECT **TO_CHAR(hire_date, 'DDD fmMMfm ddth, Year')** FROM employees;

SELECT **TO_CHAR(SYSDATE, 'DDD fmMMfm ddth, yyyy')** FROM dual;

-- Output: 322 11 17th, 2020

SELECT **TO_CHAR(SYSDATE, 'DD fmMMfm ddth, yyyy')** FROM dual;

-- Output: 17 11 17th, 2020

- Time Display with AM/PM and 24-Hour Clock

```
SELECT TO_CHAR(SYSDATE, 'HH24-MI-SS PM - mm, dd, yyyy') FROM dual;

-- Output: 10-49-11 AM - 11, 17, 2020


SELECT TO_CHAR(SYSDATE, 'HH-MI-SS - mm, dd, yyyy') FROM dual;

-- Output: 10-50-44 - 11, 17, 2020
```

*** Even if you use AM/PM, the system time will reflect actual system clock.*

- TO_CHAR(date, 'format') with Custom Text

```
SELECT TO_CHAR(SYSDATE, 'DD "OF" MONTH') FROM dual;

-- Output: 17 OF NOVEMBER
```

***You can include any static characters inside double quotes within a format string.*

🟧 Number → Character Formatting

- Format: '9'

```
SELECT TO_CHAR(1234, '99999') FROM dual;

-- Output: 1234
```

*** 9 is a digit placeholder that doesn't add leading zeros.*

- Format: '0'

```
SELECT TO_CHAR(1234, '099999') FROM dual;

-- Output: 001234
```

*** 0 adds leading zeros — 1 zero adds 2 digits, 2 zeros add 4, and so on.*

- **Format: '$'**

```
SELECT TO_CHAR(1234, '$099,999') FROM dual;
-- Output: $001,234
```

*** Includes a dollar sign and comma formatting.*

- **Format: 'L' (Local currency)**

```
SELECT TO_CHAR(1234, 'L99,99') FROM dual;
-- Output: $12,34 (depending on NLS settings)
SELECT TO_CHAR(1234.333, 'L99,99') FROM dual;
```

*** Might fail or round depending on decimals & width.*

- **Format: 'MI' (Negative after number)**

```
SELECT TO_CHAR(-1234, '999,9MI') FROM dual;
-- Output: 123,4-
```

- **Format: 'PR' (Negative in angle brackets)**

```
SELECT TO_CHAR(-1234, '99999PR') FROM dual;
-- Output: <1234>
```

- **Format: 'V' (Virtual decimal)**

```
SELECT TO_CHAR(4.5, '9999999V99') FROM dual;
-- Output: 000000450
```

*** Moves the decimal: 4.5 → 450*

- Format: 'B' (Blank for zero)

```
SELECT TO_CHAR(040.00, 'B999999999') FROM dual;
-- Output:   40
```

*** Suppresses leading zeros.*

🟧 Character → Date Conversion

- TO_DATE('string', 'format')

```
SELECT TO_DATE('April 07, 2020', 'Month DD, YYYY') FROM dual;
SELECT TO_DATE('April 07, 2020', 'Mon DD, YYYY') FROM dual;
```

*** Both Month and Mon are accepted.*

- Using Fx Modifier (Format-Exact)

```
SELECT TO_DATE('Apr 07, 2020', 'FxMon DD, YYYY') FROM dual;
SELECT TO_DATE('April 07, 2020', 'FxMonth DD, YYYY') FROM dual;
```

*** Ensures strict matching (Apr ≠ April in FxMon)*

- RR vs YY in Year Format

```
SELECT TO_DATE('June 19, 04', 'Month DD, YY') FROM dual;
-- Output: 19-JUN-2004
SELECT TO_DATE('27-OCT-95', 'DD-Mon-YY') FROM dual;
-- Output: 27-OCT-2095 ❌
```

(It gives us 2095 'In this case' we'll use RR instead of YY)

```
SELECT TO_DATE('27-OCT-95', 'DD-Mon-RR') FROM dual;
-- Output: 27-OCT-1995 ☑
```

Current Year	Specified Date	RR Format	YY Format
1995	27-Oct-95	1995	1995
1995	27-Oct-17	2017	1917
2017	27-Oct-17	2017	2017
2017	27-Oct-95	1995	2095

🟧 CH7: NULL Handling & Conditional Value Functions

- **NVL(expr1, expr2)**

If expr1 is NULL, returns expr2. Otherwise, returns expr1.

Select **nvl** (Salary, 0) from employees;

--> Null values will be displayed as 0's

Select **nvl**(Date_of_independents , 'No Date') from wf_country;

--> All null Dates will be Displayed as No date .

Other examples:

Select last_name,**NVL** (Commetion_pct,0)*250 AS "commission"

from employees Where department_id IN(80,90);

- **NVL2(expr1, expr2, expr3)**

Logic:

- If expr1 is **not NULL** → returns expr2
- If expr1 is **NULL** → returns expr3

SELECT last_name, salary,

NVL2(**commission_pct**, salary + 5, salary) AS income FROM employees;

➔ *Adds 5 to salary only if commission exists. Else salary will be the same.*

- NULLIF(expr1, expr2)

** Returns NULL if both expressions are equal; otherwise returns expr1.*

```
SELECT first_name, LENGTH(first_name) AS "Length FN",
last_name, LENGTH(last_name) AS "Length LN",
NULLIF(LENGTH(first_name), LENGTH(last_name)) AS "Compare Them"
FROM employees;
```

Test example:

```
SELECT NULLIF(LENGTH('Hind'), LENGTH('Hind222')) FROM dual;
-- Output: NULL (lengths are not equal)
```

- COLALESCE FUNCTION :

** Returns the first non-NULL expression in the list.*

```
SELECT last_name,
COALESCE(commission_pct, salary, 10) AS "Comm"
FROM employees
ORDER BY commission_pct;

Select COAlEsce(commission_pct,First_Name, Last_Name )
From employees;
-- Returns the first non-null value among the three columns.
```

FUNCTION	DESCRIPTION
NVL	Replace NULL with a default value
NVL2	Choose between two values depending on whether input is null
NULLIF	Compare two values; return NULL if equal
COALESCE	Return the first non-null value from a list

🟧 Conditional Expressions – CASE Statement:

```
CASE expr
   WHEN comparison_expr1 THEN return_expr1
   WHEN comparison_expr2 THEN return_expr2
   ...
   ELSE else_expr
END
```

Example:

```
SELECT last_name,department_id,
CASE department_id WHEN 90 THEN 'Management'
        WHEN 80 THEN 'Sales'
        WHEN 60 THEN 'It'
        ELSE 'Other dept.'
END AS "Department" FROM employees;
```

🟧 CH7: Joins in SQL

---

1. NATURAL JOIN

* Automatically joins tables based on columns with the same name and data type
* No need to specify the join condition manually.

```
SELECT first_name, last_name, job_title
FROM employees
NATURAL JOIN jobs
WHERE department_id > 80;
```

Another example:

```
SELECT department_name, city
FROM departments
NATURAL JOIN locations;
```

---

2. CROSS JOIN

Join each row of a table to every row in the other table.

* First Row of table1 with all the rows from table2
* Second Row of table1 with all the rows from table2, And so on ..

```
SELECT last_name, department_name
FROM employee
CROSS JOIN department;
```

```
SELECT tname, person_id
FROM teacherhct
CROSS JOIN personsal;

SELECT first_name, job_title
FROM employees
CROSS JOIN jobs;
```

➔ If table A has 4 rows and table B has 3 → result = 4×3 = 12 rows.

---

3. JOIN ... USING(column_name)

Explicitly joins using a common column that has the same name in both tables.

```
SELECT first_name, last_name, department_id, department_name
FROM employees
JOIN departments USING (department_id);
```

- **With condition:**

```
SELECT first_name, last_name, department_id, department_name
FROM employees
JOIN departments USING (department_id)
WHERE department_id = 10;
```

---

4. Aliases + JOIN / WHERE

```
SELECT e.first_name, e.last_name, j.job_id
FROM employees e
JOIN jobs j USING (job_id);
```

- Using traditional JOIN with WHERE:

```
SELECT last_name, e.job_id, job_title
FROM employees e , jobs j
WHERE e.job_id = j.job_id  AND department_id = 80;
```

Another format:

```
SELECT last_name, emp.job_id, job_title AS "ddd"
FROM employees emp, jobs j
WHERE emp.job_id = j.job_id  AND department_id = 80;
```

🟧 CH7.1: ON Clause in SQL Joins

- When to Use ON

- Use when **column names differ** between the tables.
- Use when the join needs **non-equality** conditions like <, >, BETWEEN.
- Unlike USING, ON allows **flexibility** in join logic.
- You can still apply **WHERE** conditions afterward.

- **Syntax:**

SELECT column_list

FROM table1 **alias1**

JOIN table2 **alias2**

ON (alias1.column = alias2.column);

- **Example from the slide:**

SELECT last_name, job_title

FROM **employees e**

JOIN jobs j

ON (e.job_id = j.job_id);

*** Required when job_id in employees ≠ job_id in jobs by name or if you need complex conditions.*

🔸 Q1. Locations + Countries Join

- Show capitalized city, and replace 'a' with '$' in country_name
  Condition:
  - location_id > 500
  - city ends with 'e'

Answer:

SELECT INITCAP(city), REPLACE(country_name, 'a', '$')

FROM locations

NATURAL JOIN countries

WHERE location_id > 500 AND city LIKE '%e';

Q2. Employee Name Format with @

- Display employee_id, and first_name with e replaced by @
  Label the name column as **"EMPLOYEE NAME"**
  Exclude IDs 103 and 100

Answer:

```
SELECT employee_id,
    REPLACE(first_name, 'e', '@') AS "EMPLOYEE NAME"
FROM employees
WHERE employee_id NOT IN (103, 100);
```

Q3. Email + Password Generation (Advanced)

Use Employees table to generate:

- Full name (First + Last)
- Email: lowercase first_name + @hct.edu.om
- Password: last 3 letters of first_name, starting with a capital letter

Answer:

```
SELECT first_name,
    last_name,
    LOWER(first_name) || '@hct.edu.om' AS "Email Address",
    INITCAP(SUBSTR(first_name, -3)) AS "Password"
FROM employees;
```

*** Explanation:*

LOWER(first_name) → lowercase email prefix

SUBSTR(first_name, -3) → last 3 characters

INITCAP(...) → capitalize first letter of password

🟧 CH7.2: Group Functions in SQL

- SUM

SELECT **SUM**(salary)

FROM employees;

** Adds up all salary values.*

- AVG (Average)

SELECT **AVG**(salary)

FROM employees

WHERE department_id = 10;

** Average salary in department 10.*

SELECT **ROUND**(**AVG**(salary), **2**)

FROM employees

WHERE department_id = 90;

** Rounded average to 2 decimal places.*

🔹 COUNT

SELECT **COUNT**(*) FROM employees;

-- Counts all rows including NULLs

SELECT **COUNT**(hire_date) FROM employees;

-- Counts only rows with non-null hire_date

SELECT **COUNT**(job_id) FROM employees;

SELECT **COUNT**(commission_pct) FROM employees;

** Ignores NULL values.*

- ## VARIANCE & STDDEV

SELECT **VARIANCE**(salary) FROM employees;

SELECT **STDDEV**(salary) FROM employees;

** Measures how spread out salary values are — standard deviation & variance.*
** Helps understand deviation from the average.*

- ## MAX, MIN (Multiple Group Functions)

SELECT **MAX**(salary), **MIN**(salary), **MIN**(employee_id)

FROM employees

WHERE department_id = 6;

**** we cant use do :**

SELECT max(salary), min(salary), min (employee_id),employee_id

From employees

Where department_id = 6

❌ *You cannot mix group functions with individual row columns unless using GROUP BY.*

🟧 COUNT with DISTINCT

- COUNT(*)

** Counts all rows — even with nulls.*

```
SELECT COUNT(*) FROM employees;

-- Output: total number of rows
```

- COUNT(column)

** Counts only non-null values.*

```
SELECT COUNT(commission_pct) FROM employees;

-- Output: 4 (if 6 are null)
```

- COUNT(DISTINCT column)

** Counts non-duplicate, non-null values.*

```
SELECT COUNT(DISTINCT job_id)

FROM employees;
```

- DISTINCT Keyword

** Removes duplicate values from result set.*

```
SELECT DISTINCT job_id

FROM employees;


SELECT DISTINCT hire_date

FROM employees

ORDER BY hire_date;
```

🟧 CH8: GROUP BY & HAVING

- **GROUP BY Clause**

Groups rows based on one or more columns, so you can apply aggregate functions (MAX, COUNT) to each group.

Examples

```
SELECT MAX(salary)
FROM employees
GROUP BY department_id;
```

Returns the highest salary per department.

```
SELECT MAX(salary), department_id
FROM employees
GROUP BY department_id;
```

Now we see both the department ID and its max salary.

- **Another grouping example:**

```
SELECT MAX(salary)
FROM employees
GROUP BY manager_id;
```

Try this to see max salary per manager ID.

Note:

- You **can't use WHERE** to filter grouped values, that's what HAVING is for.
- WHERE filters **rows before grouping**, while HAVING filters **groups after aggregation**.

- **HAVING Clause**

Clause	Filters What?	Used With
WHERE	Individual rows (before grouping)	All queries
HAVING	Grouped results (after GROUP BY)	Only with GROUP BY

Examples

SELECT department_id, job_id, COUNT(*)

FROM employees

GROUP BY department_id, job_id

HAVING COUNT(*) > 2;

** Shows departments & jobs that appear more than twice.*

SELECT department_id, job_id, COUNT(*)

FROM employees

GROUP BY department_id, job_id

HAVING department_id > 50;

** Filters grouped rows where department_id > 50.*

- **More Examples:**

```
SELECT department_id, MAX(salary)
FROM employees
GROUP BY department_id
HAVING COUNT(*) > 1
ORDER BY department_id;
```

```
SELECT department_id, MAX(salary)
FROM employees
GROUP BY department_id
HAVING MAX(salary) = 5800
ORDER BY department_id;
```

** This shows only departments where the highest salary = 5800.*

🟧 CH9: Subqueries and Multi-Row Comparison Operators

What is a Subquery?

A subquery is a query nested inside another query's WHERE, FROM, or SELECT.

Example: Simple Subquery with >

```
SELECT first_name, last_name, hire_date  FROM employees
WHERE hire_date > (
   SELECT hire_date
   FROM employees
   WHERE last_name = 'Vargas'  );
```

** Returns employees hired after Vargas.*

1. ANY Operator

** Compares a value to any value returned by the subquery.*
** Return rows if the condition is true for at least one value.*

```
SELECT salary
FROM employees
WHERE salary > ANY (
    SELECT salary
    FROM employees
    WHERE department_id = 20
);
```

~ If department 20 has salaries 3000, 3500, 4000 →
It returns salaries higher than at least one of these (i.e., >3000).

---

2. ALL Operator

* Compares a value to **all values** returned by the subquery.

* Condition must be true for every value in the result set.

```
SELECT salary
FROM employees
WHERE salary > ALL (
    SELECT salary
    FROM employees
    WHERE department_id = 20
);
```

Only returns salaries higher than everyone in dept 20.

3. IN Operator

* Matches exact values from the subquery.

```
SELECT last_name, employee_id
FROM employees
WHERE employee_id IN (
    SELECT manager_id
    FROM employees
);
```

* Returns employees who are managers (since their IDs appear as manager_id).

Example2:

```
SELECT first_name, last_name, hire_date
FROM employees
WHERE first_name IN (
    SELECT first_name
    FROM employees
    WHERE first_name LIKE '%a%'
);
```

➔ Finds employees whose first names contain 'a' — since the inner query collects all such names.

🟧 CH10: SQL Views

- What is a View?

A view is a virtual table based on a SQL query.
It can:

- Show specific columns from a table
- Apply restrictions to limit visible data
- Be updated like a table (with some rules)

---

1. Create a View

```
CREATE VIEW view_employees AS
SELECT employee_id, first_name, last_name, email, phone_number
FROM employees;
```

* Creates a view named view_employees with selected columns only.

```
SELECT * FROM view_employees;
```

---

2. Create or Replace View

```
CREATE OR REPLACE VIEW view_employees AS
SELECT employee_id, first_name, last_name
FROM employees;
```

* Replaces the view if it already exists, with a new column set.

- Simple View

- Has One base table
- No functions, group by, joins, etc.

We *can also insert into simple views:*

INSERT INTO view_employees VALUES (123, 'Ali', 'Ahmed');

* Works only if all columns are from one base table and no constraints are violated.

- Complex View has:

- Joins
- Group by
- Aggregates (e.g., MIN, MAX)

CREATE **VIEW** view_123 AS

SELECT employee_id, job_id, job_title

FROM employees

JOIN jobs **USING** (job_id);

~ This view cannot be updated/inserted directly.

Feature	Simple Views	Complex Views
Number of tables used to derive data	One	One or more
Can contain functions	No	Yes
Can contain groups of data	No	Yes
Can perform DML operations (INSERT, UPDATE, DELETE) through a view	Yes	Not always

🟧 CH11.1: GRANT – Giving Privileges

---

1. **Grant SELECT to All Users (PUBLIC)**

GRANT SELECT ON view_employees TO PUBLIC;

~ Gives read access to all users.

2. To view the shared data:

SELECT * FROM **username**.tablename;

SELECT * FROM **om_a835_sql_s57**.view_employees;

---

3. Grant SELECT to a Specific User

GRANT SELECT ON view_employees TO **om_a102_sql_s40**;

~ Now only that user can read the view.

---

4. Grant UPDATE Privileges

GRANT UPDATE ON view_employees TO **PUBLIC**;

GRANT UPDATE ON **view_employees** TO om_a102_sql_s40;

~ Cannot update complex views, only simple views.

---

5. Grant DELETE Privileges

GRANT DELETE ON view_employees TO PUBLIC;

GRANT DELETE ON **view_employees** TO om_a102_sql_s40;

---

6. Grant ALL Privileges (Insert, Select, Update, Delete)

GRANT INSERT, SELECT, UPDATE, DELETE

ON view_employees TO **PUBLIC**;

🟧 CH11.2: REVOKE – Removing Privileges

---

1. **Revoke SELECT**

REVOKE SELECT ON view_employees FROM **PUBLIC**;

---

2. **Revoke UPDATE**

REVOKE UPDATE ON view_employees FROM PUBLIC;

REVOKE UPDATE ON view_employees FROM **om_a102_sql_s40**;

---

3. **Revoke DELETE**

REVOKE DELETE ON view_employees FROM PUBLIC;

REVOKE DELETE ON view_employees FROM **om_a102_sql_s40**;

---

4. **Revoke ALL Privileges**

REVOKE INSERT, SELECT, UPDATE, DELETE

ON view_employees FROM **PUBLIC**;

🟧 CH11.2: Transaction Control in SQL

COMMAND	DESCRIPTION
SAVEPOINT T1	Marks a point to return to
ROLLBACK T1	Undoes changes after T1
COMMIT	Finalizes all changes, cannot rollback

Example of savepoint:

At Step 3 – Update Ford to BMW → SAVEPOINT T1

```
UPDATE car
SET carname = 'BMW'
WHERE carno = 200;
SAVEPOINT T1;
```

At Step 4 – Insert Lexus → SAVEPOINT T2

```
INSERT INTO car VALUES(300, 'Lexus');
SAVEPOINT T2;
```

At Step 5 – Update Honda to GMC → SAVEPOINT T3

```
UPDATE car
SET carname = 'GMC'
WHERE carno = 100;
SAVEPOINT T3;
```

Example of ROLLBACK:

```
ROLLBACK TO T2;
```

- **Keeps** changes up to T2 (BMW and Lexus insert stay)
- **Undoes** the update to GMC and deletion of car 200

- **COMMIT:**

```
COMMIT;
```

~ Locks in all changes made before — no rollback possible after this.