Master SQL Cheat Sheet — Full Version (Professional)

Sample database: company_db

Dummy Tables

Table: employees

emp_id	name	dept_id	salary	hire_date
1	Alice	10	70000	2018-02-01
2	Bob	10	50000	2019-05-10
3	Charlie	20	65000	2020-07-18
4	David	20	45000	2021-09-25
5	Emma	30	80000	2017-03-05
6	Frank	20	65000	2022-01-15
7	Grace	30	72000	2019-11-11
8	Hannah	10	70000	2020-12-01

Table: departments

dept_id	dept_name
10	HR
20	IT
30	Finance

Table: projects

proj_id	proj_name	lead_emp_id
1	Alpha	3
2	Beta	2
3	Gamma	4
4	Delta	1

SELECT all employees

Query:

```
SELECT * FROM employees;
```

Result:

emp_id	name	dept_id	salary	hire_date
1	Alice	10	70000	2018-02-01
2	Bob	10	50000	2019-05-10
3	Charlie	20	65000	2020-07-18
4	David	20	45000	2021-09-25
5	Emma	30	80000	2017-03-05
6	Frank	20	65000	2022-01-15
7	Grace	30	72000	2019-11-11
8	Hannah	10	70000	2020-12-01

Filter: salary > 70000

Query:

```
SELECT name, salary FROM employees WHERE salary > 70000;
```

Result:

name	salary
Emma	80000
Grace	72000

Group & Having: dept avg > 65000

Query:

```
SELECT dept_id, AVG(salary) AS avg_salary FROM employees GROUP BY dept_id HAVING AVG(salary) > 65000;
```

Result:

dept_id	avg_salary
30.0	76000.0

Join: employees with departments

Query:

```
SELECT e.emp_id, e.name, d.dept_name, e.salary FROM employees e INNER JOIN departments d ON e.dept_id = d.dept_id;
```

Result:

emp_id	name	dept_name	salary
1	Alice	HR	70000
2	Bob	HR	50000
3	Charlie	IT	65000
4	David	IT	45000
5	Emma	Finance	80000
6	Frank	IT	65000
7	Grace	Finance	72000
8	Hannah	HR	70000

Subquery: highest paid employee(s)

Query:

```
SELECT name, salary FROM employees WHERE salary = (SELECT MAX(salary) FROM employees);
```

Result:

name	salary
Emma	80000

Correlated Subquery: employees earning > dept avg

Query:

```
SELECT e.name, e.dept_id, e.salary FROM employees e WHERE e.salary > (SELECT AVG(salary) FROM employees WHERE dept_id = e.dept_id);
```

Result:

name	dept_id	salary
Alice	10	70000
Charlie	20	65000
Emma	30	80000
Frank	20	65000
Hannah	10	70000

Window: ROW_NUMBER partition by dept

Query:

```
SELECT name, dept_id, salary, ROW_NUMBER() OVER (PARTITION BY dept_id ORDER BY salary DESC) AS row_no FROM employees;
```

Result:

name	dept_id	salary	row_number
Alice	10	70000	1
Hannah	10	70000	2
Bob	10	50000	3
Charlie	20	65000	1
Frank	20	65000	2
David	20	45000	3
Emma	30	80000	1
Grace	30	72000	2

Window: RANK and DENSE_RANK over salary desc

Query:

```
SELECT name, salary, RANK() OVER (ORDER BY salary DESC) AS rnk, DENSE_RANK() OVER (ORDER BY salary DESC) AS dense_rnk FROM employees;
```

Result:

name	salary	rank	dense_rank
Emma	80000	1	1
Grace	72000	2	2
Alice	70000	3	3

name	salary	rank	dense_rank
Hannah	70000	3	3
Charlie	65000	5	4
Frank	65000	5	4
Bob	50000	7	5
David	45000	8	6

NTILE(3) over salary desc

Query:

```
SELECT name, salary, NTILE(3) OVER (ORDER BY salary DESC) AS ntile FROM employees;
```

Result:

name	salary	ntile
Emma	80000	1
Grace	72000	1
Alice	70000	1
Hannah	70000	2
Charlie	65000	2
Frank	65000	3
Bob	50000	3
David	45000	3

LAG & LEAD by hire_date

Query:

```
SELECT name, hire_date, LAG(salary) OVER (ORDER BY hire_date) AS prev_salary, LEAD(salary) OVER (ORDER BY hire_date) AS next_salary FROM employees;
```

Result:

name	hire_date	prev_salary	next_salary
Emma	2017-03-05	nan	70000.0
Alice	2018-02-01	80000.0	50000.0
Bob	2019-05-10	70000.0	72000.0
Grace	2019-11-11	50000.0	65000.0
Charlie	2020-07-18	72000.0	70000.0
Hannah	2020-12-01	65000.0	45000.0
David	2021-09-25	70000.0	65000.0
Frank	2022-01-15	45000.0	nan

CTE: employees earning > dept avg (CTE example)

Query:

```
WITH dept_avg AS (SELECT dept_id, AVG(salary) AS avg_sal FROM employees GROUP BY dept_id) SELECT e.name, e.salary FROM employees e JOIN dept_avg d ON e.dept_id = d.dept_id WHERE e.salary > d.avg_sal;
```

Result:

name	dept_id	salary	avg_sal
Alice	10	70000	63333.33
Charlie	20	65000	58333.33
Emma	30	80000	76000.0
Frank	20	65000	58333.33
Hannah	10	70000	63333.33

Top 2 earners per department

Query:

```
SELECT name, dept_id, salary FROM (SELECT name, dept_id, salary, ROW_NUMBER() OVER (PARTITION BY dept_id ORDER BY salary DESC) AS rn FROM employees) t WHERE rn <= 2;
```

Result:

name	dept_id	salary	row_number
Alice	10	70000	1
Hannah	10	70000	2
Charlie	20	65000	1
Frank	20	65000	2
Emma	30	80000	1
Grace	30	72000	2

Self join: employee with manager name (demo)

Query:

```
SELECT e.emp_id, e.name, m.name AS manager_name FROM employees e LEFT JOIN employees m ON e.emp_id = m.emp_id /* replace with manager mapping */;
```

Result:

emp_id	name	manager_name
1	Alice	nan
2	Bob	Alice
3	Charlie	Alice
4	David	Charlie
5	Emma	David
6	Frank	nan
7	Grace	nan
8	Hannah	nan

Recursive CTE Example (Generate sequence 1..5)

Query:

```
WITH RECURSIVE nums AS (SELECT 1 AS n UNION ALL SELECT n+1 FROM nums WHERE n < 5) SELECT * FROM nums;
```

Result:

n
1

2
3
4
5

Interview Questions — Subqueries & Window Functions

Q1: Find employees who earn more than the company average.

Query:

```
SELECT name, salary FROM employees WHERE salary > (SELECT AVG(salary) FROM employees);
```

Result:

name	salary
Alice	70000
Charlie	65000
Emma	80000
Frank	65000
Grace	72000
Hannah	70000

Q2: For each department, find the employee(s) with the highest salary.

Query:

```
SELECT name, dept_id, salary FROM (SELECT name, dept_id, salary, RANK() OVER(PARTITION BY dept_id ORDER BY salary DESC) AS rnk FROM employees) t WHERE rnk = 1;
```

Result:

name	dept_id	salary
Alice	10	70000
Charlie	20	65000
Emma	30	80000
Frank	20	65000
Hannah	10	70000

Q3: Find the second highest salary in the company.

Query:

```
SELECT MAX(salary) FROM employees WHERE salary < (SELECT MAX(salary) FROM employees);
```

Result:

metric	value
second_highest	72000

Q4: Running total of salaries ordered by hire_date.

Query:

```
SELECT name, hire_date, SUM(salary) OVER (ORDER BY hire_date ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) AS running_total FROM employees;
```

Result:

name	hire_date	running_total
Emma	2017-03-05	80000
Alice	2018-02-01	150000
Bob	2019-05-10	200000
Grace	2019-11-11	272000

name	hire_date	running_total
Charlie	2020-07-18	337000
Hannah	2020-12-01	407000
David	2021-09-25	452000
Frank	2022-01-15	517000

Q5: Assign employees into 4 salary quartiles using NTILE(4).

Query:

```
SELECT name, salary, NTILE(4) OVER (ORDER BY salary DESC) AS quartile FROM employees;
```

Result:

name	salary	quartile
Emma	80000	1
Grace	72000	1
Alice	70000	2
Hannah	70000	2
Charlie	65000	3
Frank	65000	3
Bob	50000	4
David	45000	4

Performance & Interview Tips (Summary)

- Use EXPLAIN/EXPLAIN ANALYZE to inspect query plans.
- Prefer indexed joins on large tables; avoid functions on joined columns.
- For large datasets, use window functions instead of correlated subqueries when possible.
- Practice writing both set-based and window-based solutions — interviewers value both.