Oracle 1Z0-071 Cheat Sheet (Sections 6-12)

6. Reporting Aggregated Data Using Group Functions

COUNT(*)	- Total number of rows
COUNT(col)	- Non-null values only
SUM(col)	- Total of numeric values
AVG(col)	- Average of values
MIN(col)	- Minimum value
MAX(col)	- Maximum value

GROUP BY Example:

```
SELECT department_id, COUNT(*)
FROM employees
GROUP BY department_id;
```

HAVING Clause:

```
SELECT department_id, COUNT(*)
FROM employees
GROUP BY department_id
HAVING COUNT(*) > 5;
```

7. Displaying Data from Multiple Tables Using Joins

INNER JOIN:

```
SELECT e.first_name, d.department_name
FROM employees e
JOIN departments d ON e.department_id = d.department_id;
```

OUTER JOIN:

```
SELECT e.first_name, d.department_name
FROM employees e
LEFT JOIN departments d ON e.department_id = d.department_id;
```

USING Clause:

```
SELECT e.first_name, d.department_name
FROM employees e
JOIN departments d USING (department_id);
```

SELF JOIN:

```
SELECT e1.first_name, e2.first_name AS manager
FROM employees e1
JOIN employees e2 ON e1.manager_id = e2.employee_id;
```

8. Using Subqueries to Solve Queries

Single-Row Subquery:

```
SELECT first_name
FROM employees
WHERE salary > (SELECT AVG(salary) FROM employees);
```

Multi-Row Subquery:

```
SELECT first_name
FROM employees
WHERE department_id IN (
  SELECT department_id FROM departments WHERE location_id = 1700
);
```

Correlated Subquery:

```
SELECT e1.first_name
FROM employees e1
WHERE salary > (
  SELECT AVG(salary)
  FROM employees e2
  WHERE e1.department_id = e2.department_id
);
```

EXISTS:

```
SELECT department_name
FROM departments d
WHERE EXISTS (
  SELECT 1 FROM employees e WHERE e.department_id = d.department_id
);
```

9. Using SET Operators

```
UNION      - Combines results, removes duplicates
UNION ALL  - Combines all, keeps duplicates
INTERSECT  - Rows common to both queries
MINUS      - Rows in first but not in second
```

Example:

```
SELECT employee_id FROM employees
UNION
SELECT employee_id FROM job_history;
```

10. Manipulating Data

INSERT:

```
INSERT INTO employees (employee_id, first_name) VALUES (300, 'John');
```

UPDATE:

```
UPDATE employees SET salary = salary * 1.1 WHERE department_id = 50;
```

DELETE:

```
DELETE FROM employees WHERE employee_id = 300;
```

Transaction Control:

```
COMMIT       - Save changes permanently
ROLLBACK     - Undo changes
SAVEPOINT sp1;
ROLLBACK TO sp1;
```

11. Using DDL Statements to Create and Manage Tables

CREATE TABLE:

```
CREATE TABLE departments (
  dept_id NUMBER PRIMARY KEY,
  name VARCHAR2(50) NOT NULL
);
```

ALTER TABLE:

```
ALTER TABLE departments ADD (location_id NUMBER);
ALTER TABLE departments MODIFY (name VARCHAR2(100));
ALTER TABLE departments DROP COLUMN location_id;
```

DROP TABLE:

```
DROP TABLE departments;
```

Constraints:
PRIMARY KEY, FOREIGN KEY, NOT NULL, UNIQUE, CHECK

```
ALTER TABLE employees ADD CONSTRAINT emp_dept_fk
FOREIGN KEY (department_id) REFERENCES departments(dept_id);
```

12. Creating Other Schema Objects

VIEW:

```
CREATE VIEW emp_view AS
SELECT first_name, salary FROM employees;
```

SEQUENCE:

```
CREATE SEQUENCE emp_seq START WITH 100 INCREMENT BY 1;
```

INDEX:

```
CREATE INDEX emp_name_idx ON employees (last_name);
```

SYNONYM:

```
CREATE SYNONYM emp FOR employees;
```