SQL Functions and Commands Summary

1. Data Types

- VARCHAR(n): Variable-length string (up to 65,535 bytes)
- CHAR(n): Fixed-length string, padded with spaces
- INT: Whole number (-2,147,483,648 to 2,147,483,647)
- DECIMAL(p,s): Fixed-point number with precision and scale
- FLOAT(p,d): Approximate floating-point number
- DOUBLE: Larger floating-point number than FLOAT
- DATE: Date in YYYY-MM-DD format
- DATETIME: Date and time in YYYY-MM-DD HH:MM:SS format
- TIMESTAMP: Time zone-aware date/time, can auto-update
- BINARY(n): Fixed-length binary data
- BLOB: Binary data (up to 65,535 bytes)

2. CREATE TABLE Statement

Used to create a new table with columns, datatypes, and constraints.

Syntax:

```
CREATE TABLE table_name (
    column1 datatype [constraint],
    column2 datatype [constraint],
    ...
    [table_constraints]
);
```

3. Integrity Constraints

SQL Functions and Commands Summary

- NOT NULL: Ensures a column cannot be NULL
- UNIQUE: No duplicate values allowed (NULLs allowed)
- PRIMARY KEY: Uniquely identifies a row (NOT NULL + UNIQUE)
- CHECK (condition): Ensures values meet a condition
- DEFAULT: Provides default value for a column
- AUTO_INCREMENT: Automatically increases numeric values (IDs)

4. Foreign Key (Referential Integrity)

Maintains data consistency between related tables.

Syntax:

```
FOREIGN KEY (column) REFERENCES parent_table(column)
   ON DELETE SET NULL
   ON UPDATE CASCADE;
```

5. Table Constraints Syntax

Column-Level:

```
column_name datatype CONSTRAINT constraint_name constraint_type
```

Table-Level:

```
CONSTRAINT constraint_name PRIMARY KEY (col1, col2),
FOREIGN KEY (col) REFERENCES table(col),
CHECK (condition)
```

6. DROP TABLE

Deletes an entire table and its definition.

Syntax:

```
DROP TABLE table_name;
```

7. ALTER TABLE

Modify structure of existing tables.

- Add column:

```
ALTER TABLE table_name ADD column_name datatype;
```

- Drop column:

```
ALTER TABLE table_name DROP COLUMN column_name CASCADE;
```

- Drop default constraint:

```
ALTER TABLE table_name ALTER column_name DROP DEFAULT;
```

8. INSERT INTO

Adds data rows to a table.

Syntax:

```
INSERT INTO table_name (column1, column2, ...)
VALUES (value1, value2, ...);
```

SQL Queries and SELECT Statement Summary

1. Basic SELECT Statement

- SELECT <attribute list> FROM <table list> WHERE <condition>;
- Use * to select all columns.
- Use column aliases for clarity.
- SQL tables allow duplicate rows (multi-set).
- Use DISTINCT to eliminate duplicates.

2. Arithmetic Operations

Example:

SELECT last_name, salary, 12*(salary+100) FROM emps;

- Can use +, -, *, / with parentheses.

3. Using Column Aliases

Example:

SELECT last_name AS "Name", salary*12 AS "Annual Salary" FROM emps;

4. Simple SQL Queries

- SELECT: retrieves rows (selection).
- PROJECT: retrieves specific columns.
- JOIN: combines rows from multiple tables.

Example:

SQL Queries and SELECT Statement Summary

SELECT BDATE, ADDRESS FROM EMPLOYEE WHERE FNAME='John' AND LNAME='Smith';

5. Joins with Two or More Tables

Example with two tables:

SELECT FNAME, LNAME, ADDRESS FROM EMPLOYEE, DEPARTMENT

WHERE DNAME='Research' AND DNUMBER=DNO;

Example with three tables:

SELECT PNUMBER, DNUM, LNAME, BDATE, ADDRESS

FROM PROJECT, DEPARTMENT, EMPLOYEE

WHERE DNUM=DNUMBER AND MGRSSN=SSN AND PLOCATION='Stafford';

6. Aliasing Tables

Example:

SELECT E.FNAME, E.LNAME, S.FNAME, S.LNAME

FROM EMPLOYEE E, EMPLOYEE S

WHERE E.SUPERSSN = S.SSN;

- Aliases (E, S) are used for clarity or when referencing the same table multiple times.

7. Empty WHERE-Clause and * Usage

- No WHERE clause = all rows are selected.
- Example: SELECT SSN FROM EMPLOYEE;
- * selects all attributes: SELECT * FROM EMPLOYEE WHERE DNO=5;

SQL Queries and SELECT Statement Summary

8. Using DISTINCT

- Removes duplicate rows.

Example:

```
SELECT DISTINCT SALARY FROM EMPLOYEE;
```

9. Practice Activities

1. Display last name, weekly salary, and department number with alias "Weekly Salary".
2. Find employees supervised by 'Franklin Wong'.
3. Find female employees with a dependent of the same first name.
4. For each department, get manager's last name, start date, and dependent names.
5. For each employee, get name, department, project, and hours worked.

SQL Functions and Code Examples - CSE 311L Week 3

1. SELECT with WHERE Clause

Used to retrieve data from specific rows that match a condition.

```
SELECT employee_id, last_name, job_id FROM emps WHERE department_id = 90;
```

2. Character Strings and Date Filtering

Filter rows using specific string or date values.

```
SELECT last_name, job_id FROM emps WHERE last_name = 'WHALEN';
```

3. Comparison Operators

Use =, <, >, <=, >= to compare values.

```
SELECT last_name, salary FROM emps WHERE salary <= 3000;
```

4. BETWEEN and IN

BETWEEN for range, IN for matching multiple values.

```
SELECT salary FROM emps WHERE salary BETWEEN 2500 AND 3500;
SELECT last_name FROM emps WHERE manager_id IN (100, 101, 201);
```

5. ORDER BY Clause

Sort result rows by one or more columns.

```
SELECT last_name FROM emps ORDER BY hire_date DESC;
```

6. LIKE Operator

Match partial strings using % and _ wildcards.

```
SELECT last_name FROM emps WHERE last_name LIKE '_o%';
```

7. IS NULL and IS NOT NULL

Check for missing (NULL) values.

```
SELECT last_name FROM emps WHERE manager_id IS NULL;
```

8. Logical Operators (AND, OR, NOT)

Combine multiple conditions.

```
SELECT last_name FROM emps WHERE salary >= 10000 AND job_id LIKE '%MAN%';
```

9. SET Operations (UNION, INTERSECT, MINUS)

Combine results of multiple queries.

```
(SELECT pname FROM project, department, employee WHERE lname='Wong')
UNION
(SELECT pname FROM project, works_on, employee WHERE name='Wong');
```

10. Nested Queries

Subqueries within WHERE clause.

```
SELECT fname FROM employee WHERE dno IN
(SELECT dnumber FROM department WHERE dname='Research');
```

11. Correlated Subqueries

Subquery uses value from outer query.

```
SELECT fname FROM employee e1 WHERE salary <
(SELECT salary FROM employee e2 WHERE e2.ssn = e1.superssn);
```

12. EXISTS / NOT EXISTS

Check if a subquery returns any rows.

```
SELECT fname FROM employee e1 WHERE EXISTS
(SELECT * FROM employee e2 WHERE e2.ssn = e1.superssn AND e2.salary > e1.salary);
```