Databricks SQL (DB SQL) is a serverless data warehouse on the Databricks Lakehouse Platform that lets you run all your SQL and BI applications at scale with up to 12x better price/performance, a unified governance model, open formats and APIs, and your tools of choice – no lock-in.

### CREATE TABLES

**CREATE TABLE**

```
CREATE TABLE default.sales (  
  transaction_datetime TIMESTAMP,  
  refund_datetime TIMESTAMP,  
  bank_zip INT,  
  customer_zip INT  
);
```

**CREATE VIEW**

```
CREATE VIEW mytempview AS SELECT * FROM default.sales;
```

**CREATE OR REPLACE TABLE**

```
CREATE OR REPLACE TABLE default.sales parquet.`/path/to/data`;
```

### ALTER TABLE

**RENAME TABLE**

```
RENAME TABLE sales TO salesperson;
```

**RENAME COLUMN**

```
RENAME COLUMN customer_first_name TO customer_name;
```

**ADD COLUMNS**

```
ALTER TABLE sales ADD columns (time TIMESTAMP, col_name1 data_type1);
```

**CHECK (CONSTRAINTS)**

```
-- Add a CHECK constraint
ALTER TABLE sales
ADD CONSTRAINT dateWithinRange CHECK (year > '2000-01-01');
```

**NOT NULL (CONSTRAINTS)**

```
-- Add a NOT NULL constraint
ALTER TABLE sales
ADD CONSTRAINT customer_name IS NOT NULL;
```

**DROP CONSTRAINT (CONSTRAINTS)**

```
ALTER TABLE default.sales
DROP CONSTRAINT dateWithinRange;
```

### ADD/MODIFY DATA

**UPDATE**

```
-- Update column values for rows that match a predicate
UPDATE sales
SET bank_office = 'Augusta'
WHERE employee_state = 'Maine';
```

**INSERT INTO**

```
-- Insert comma separated values directly into a table.
INSERT INTO mytable
VALUES ('Harper Bryant', 'Employee', 98101),
('Sara Brown', 'Contractor', 48109);
```

**MERGE INTO**

```
-- Upsert (update + insert) using MERGE
MERGE INTO target
USING updates
ON target.Id = updates.Id
WHEN MATCHED AND target.delete_flag = "true" THEN
  DELETE
WHEN MATCHED THEN
  UPDATE SET
WHEN NOT MATCHED THEN
  INSERT (date, Id, data) -- or, use INSERT *
VALUES (date, Id, data);
```

### COMMON SELECT QUERIES

**SUBQUERIES**

```
-- Query an intermediate result set using a subquery.
SELECT * FROM sales
WHERE sales_id IN (  
  SELECT DISTINCT sales_id FROM visit  
);
```

**ALIAS COLUMN**

```
-- Alias a column

SELECT sales_id AS sales_id_new
FROM sales;
```

**ALIAS TABLE**

```
-- Alias a table

SELECT * FROM my_sales AS m;
```

**ORDER BY**

```
-- Return a table sorted by a column's values. Values returned in ascending order by default, or specify DESC.
SELECT productname, sales_id FROM sales
ORDER BY sales_id [DESC];
```

**WHERE**

```
-- Filter a table based upon rows that match one or more specific predicates (text or numeric filtering)
SELECT * FROM sales
WHERE product_name = "Lego set" AND sales_id > 50000;
```

**JSON**

```
-- Extract values from a JSON string using the : operator, delimiters and identifiers

SELECT raw:owner, raw:OWNER, raw:['owner'], raw:['OWNER']
FROM sales;
```

```
-- Extract nested fields from JSON string using the : operator and dot notation

SELECT raw:store.bicycle
FROM sales;
```

```
-- Extract values from an array in JSON using the : operator

SELECT raw:store.fruit[0], raw:store.fruit[1]
FROM sales;
```

**CLONE**

```
-- Deep clone is a complete, independent copy of the source table
CREATE OR REPLACE TABLE default.sales DEEP CLONE parquet.`/path/to/data`;
```

```
-- Shallow clone is a copy of the source table's definition, but refers to the source table's files
CREATE OR REPLACE TABLE default.sales SHALLOW CLONE parquet.`/path/to/data`;
```
**PERMISSIONS**

- **USAGE**
  - View count of distinct records in a table
  ```sql
  SELECT COUNT(DISTINCT) sales
  FROM orderhistory;
  ```

- **AVG/MIN/MAX**
  - View average (mean), sum, min or max values in a column
  ```sql
  SELECT AVG(sales), SUM(sales), MIN(sales), MAX(sales)
  FROM orderhistory;
  ```

- **GROUP BY/HAVING**
  - View an aggregation grouped by a column's values
  ```sql
  SELECT column
  FROM Sales
  GROUP BY country
  HAVING item_type = 'soup';
  ```

**DESCRIPTION**

- **View** [detailed] information about a database or table
  ```sql
  DESCRIBE [detail] sales;
  ```

**DELTA LAKE**

- **CHANGE DATA FEED**
  - Read table changes starting at a specified version number
  ```sql
  SELECT * FROM table_changes(sales, <start version number>);
  ```

- **CONVERT TO DELTA**
  - Convert a table to Delta Lake format
  ```sql
  CONVERT TO DELTA sales;
  ```

- **VACUUM**
  - Delete files no longer used by the table from cloud storage
  ```sql
  VACUUM sales [RETAIN num hours] [DRY RUN];
  ```

**TIME TRAVEL**

- **Source**
  - Query historical versions of a Delta Lake table by version number or timestamp
  ```sql
  SELECT * FROM table_name [VERSION AS OF 0 | TIMESTAMP AS OF '2020-12-18']
  ```

**GENERAL COMMANDS**

- **Show Grant**
  - Show a user's permissions on a table
  ```sql
  SHOW GRANT 'user@example.com' ON TABLE default.sales;
  ```

**COPY INTO**

- **Create a permanent function with parameters.**
  ```sql
  CREATE FUNCTION area(x DOUBLE, y DOUBLE) RETURNS DOUBLE
  AS (area(c1, c2) > 0)
  RETURN x * y;
  ```

- **Use a SQL function in the SELECT clause of a query.**
  ```sql
  SELECT * FROM t WHERE area(c1, c2) > 0;
  ```

- **Use a SQL function in the WHERE clause of a query.**
  ```sql
  SELECT * FROM t WHERE area(c1, c2) > 0;
  ```

- **Compose SQL functions.**
  ```sql
  CREATE FUNCTION square(x DOUBLE) RETURNS DOUBLE RETURN x * x;
  ```

**COPY INTO**

- **CREATE FUNCTION**
  - Create a non-deterministic function
  ```sql
  CREATE FUNCTION roll_dice() RETURNS INT NOT DETERMINISTIC
  AS (roll_dice());
  ```

- **RETURN**
  - Roll a single 6 sided die
  ```sql
  RETURN (rand() * 6)::INT + 1;
  ```

- **RETURN**
  - Roll a single 6-sided die
  ```sql
  RETURN (rand() * 6)::INT + 1;
  ```

**DATA INGESTION**

- **COPY INTO**
  - Fileformat = JSON|CSV|PARQUET|etc.;
  ```sql
  COPY INTO /databricks-datasets/iot/*
  FROM '/databricks-datasets/iot/
  ```

- **Roll a single 6-sided die**
  ```sql
  SELECT (rand() * 6)::INT + 1;
  ```

- **Create a permanent function with parameters.**
  ```sql
  CREATE FUNCTION area(x DOUBLE, y DOUBLE) RETURNS DOUBLE
  AS (area(c1, c2) > 0)
  RETURN x * y;
  ```

- **Use a SQL function in the SELECT clause of a query.**
  ```sql
  SELECT * FROM t WHERE area(c1, c2) > 0;
  ```

- **Use a SQL function in the WHERE clause of a query.**
  ```sql
  SELECT * FROM t WHERE area(c1, c2) > 0;
  ```

- **Compose SQL functions.**
  ```sql
  CREATE FUNCTION square(x DOUBLE) RETURNS DOUBLE RETURN x * x;
  ```

- **CREATE FUNCTION**
  - Create a non-deterministic function
  ```sql
  CREATE FUNCTION roll_dice() RETURNS INT NOT DETERMINISTIC
  AS (roll_dice());
  ```

- **RETURN**
  - Roll a single 6 sided die
  ```sql
  RETURN (rand() * 6)::INT + 1;
  ```

- **RETURN**
  - Roll a single 6-sided die
  ```sql
  RETURN (rand() * 6)::INT + 1;
  ```

**INFORMATION SCHEMA**

- **View all tables that have been created in the last 24 hours**
  ```sql
  SELECT table_name, table_owner, created_by, last_altered, last_altered_by, table_catalog
  FROM system.information_schema.tables
  WHERE datadiff(now(), last_altered) < 1;
  ```

- **View how many tables you have in each schema**
  ```sql
  SELECT schema, count(table_name)
  FROM system.information_schema.tables
  WHERE table_schema = 'tpch'
  GROUP BY table_schema;
  ```

**USE**

- **Switch to a different database; the database default is used if name is specified.**
  ```sql
  USE database_name;
  ```

- **--Switch to a different database; the database default is used if none is specified.**
  ```sql
  USE default.sales;
  ```

- **--Create a common table expression (CTE) that can be easily reused in other queries.**
  ```sql
  WITH common_table_expression_name AS (CREATE FUNCTION roll_dice() RETURNS INT NOT DETERMINISTIC
  AS (roll_dice());
  ```

- **--Create a common table expression (CTE) that can be easily reused in other queries.**
  ```sql
  CREATE FUNCTION area(x DOUBLE, y DOUBLE) RETURNS DOUBLE
  AS (area(c1, c2) > 0)
  RETURN x * y;
  ```

- **Use a SQL function in the SELECT clause of a query.**
  ```sql
  SELECT * FROM t WHERE area(c1, c2) > 0;
  ```

- **Use a SQL function in the WHERE clause of a query.**
  ```sql
  SELECT * FROM t WHERE area(c1, c2) > 0;
  ```

- **Compose SQL functions.**
  ```sql
  CREATE FUNCTION square(x DOUBLE) RETURNS DOUBLE RETURN x * x;
  ```

- **CREATE FUNCTION**
  - Create a non-deterministic function
  ```sql
  CREATE FUNCTION roll_dice() RETURNS INT NOT DETERMINISTIC
  AS (roll_dice());
  ```

- **RETURN**
  - Roll a single 6 sided die
  ```sql
  RETURN (rand() * 6)::INT + 1;
  ```

- **RETURN**
  - Roll a single 6-sided die
  ```sql
  RETURN (rand() * 6)::INT + 1;
  ```

**EXPLAIN**

- **--View the physical plan for execution of a given SQL statement.**
  ```sql
  EXPLAIN [EXTENDED] SELECT * FROM sales;
  ```

- **TUNED WIDE TABLES**
  - Sets the number of columns to collect statistics on
  ```sql
  ALTER TABLE SET TBLPROPERTIES ('delta.dataSkippingNumIndexedCols' = 64);
  ```

- **OPTIMIZE**
  - OPTIMIZE Delta tables, bin packs tables for better performance
  ```sql
  OPTIMIZE sales
  ```

- **ANALYZE**
  - Optimize table to collect statistics on entire column
  ```sql
  ANALYZE TABLE sales COMPUTE STATISTICS FOR ALL COLUMNS;
  ```

- **OPTIMIZE/ZORDER**
  - Periodic OPTIMIZE and ZORDER, run on a nightly basis
  ```sql
  OPTIMIZE/TOUSER sales, ZORDER BY customer_id, customer_seq;
  ```

**CREATED FUNCTION**

- **CREATE FUNCTION**
  - Create a non-deterministic function
  ```sql
  CREATE FUNCTION roll_dice() RETURNS INT NOT DETERMINISTIC
  AS (roll_dice());
  ```

- **RETURN**
  - Roll a single 6 sided die
  ```sql
  RETURN (rand() * 6)::INT + 1;
  ```

- **RETURN**
  - Roll a single 6-sided die
  ```sql
  RETURN (rand() * 6)::INT + 1;
  ```

**GEOSPATIAL FUNCTIONS**

- **H3**
  - Returns the H3 cell ID (as a BIGINT) corresponding to the provided longitude and latitude at the specified resolution
  ```sql
  SELECT h3longlatash3(longitudeExpr, latitudeExpr, resolutionExpr)
  ```

  - Returns an ARRAY of H3 cell IDs (represented as BIGINTs) corresponding to hexagons or pentagons, of the specified resolution, that are contained by the input areal geography
  ```sql
  SELECT h3polyfillash3(geographyExpr, resolutionExpr)
  ```

  - Returns the H3 cell IDs that are within (grid) distance k of the origin cell ID
  ```sql
  SELECT h3_kring(h3CellIdExpr, kExpr)
  ```

  - Returns the grid distance of the two input H3 cell IDs
  ```sql
  SELECT h3_distance(h3CellIdExpr, h3CellId2Expr)
  ```

  - Returns the parent H3 cell ID of the input H3 cell ID at the specified resolution
  ```sql
  SELECT h3_toparent(h3CellIdExpr, resolutionExpr)
  ```