

Common SQL Data Types

INT, SMALLINT, BIGINT

Definition: Integer types with varying ranges.
Example: INT

VARCHAR(n)

Definition: Variable-length character string up to n chars. Example: VARCHAR(50)

CHAR(n)

Definition: Fixed-length character string of length n. Example: CHAR(10)

TEXT

Definition: Large variable-length text. Example: —

DATE, TIME, DATETIME, TIMESTAMP

Definition: Date and/or time values. Example: DATE

DECIMAL(p,s), NUMERIC(p,s)

Definition: Exact numeric with p total digits, s after decimal. Example: DECIMAL(10,2)

FLOAT, REAL, DOUBLE PRECISION

Definition: Approximate numeric types. Example: —

BOOLEAN

Definition: Logical TRUE/FALSE. Example: —

BLOB

Definition: Binary large object. Example: —

Joins in SQL

Inner Join

Definition: Returns rows with matches in both tables. Example:

```
SELECT *
FROM A
JOIN B ON A.x = B.x;
```

Left Outer Join

Definition: All rows from left, matched or NULLs from right. Example: A LEFT JOIN B ON A.x=B.x;

Right Outer Join

Definition: All rows from right, matched or NULLs from left. Example: A RIGHT JOIN B ON A.x=B.x;

Full Outer Join

Definition: Rows matching in either, NULLs elsewhere. Example: A FULL JOIN B ON A.x=B.x;

Self Join

Definition: A table joined to itself via aliases. Example:

```
SELECT e1.Name AS Emp, e2.Name AS Mgr
FROM Employees e1
JOIN Employees e2
ON e1.ManagerID = e2.ID;
```

Set Operations

UNION

Definition: Combines two queries, removes duplicates. Example: SELECT ... UNION SELECT ...;

UNION ALL

Definition: Combines two queries, keeps duplicates.

INTERSECT

Definition: Rows common to both, removes duplicates.

EXCEPT

Definition: Rows in first not in second.

Relational Algebra

$\sigma_C(R)$

Definition: Selection — picks rows satisfying C.
Example: $\sigma_{year>2010}(Movie)$

$\pi_{A_1, \dots, A_n}(R)$

Definition: Projection — picks columns A_i .
Example: $\pi_{name, genre}(Movie)$

$R_1 \cup R_2$

Definition: Union of two relations.

$R_1 \cap R_2$

Definition: Intersection of two relations.

$R_1 - R_2$

Definition: Set difference.

$R_1 \times R_2$

Definition: Cartesian product.

$\rho_{B_1, \dots, B_n}(R)$

Definition: Renaming relation/attributes. Example: $\rho_{M(name)}(Movie)$

$\gamma_{G;f}(R)$

Definition: Grouping + aggregate. Example: $\gamma_{genre; count(*)}(Movie)$

\bowtie_C

Definition: Theta-join = $\sigma_C(R_1 \times R_2)$. Example: see Joins box.

$\tau_{A_1, \dots, A_n}(R)$

Definition: Sorting by A_i . Example: $\tau_{year, name}(Movie)$

Aggregation & Grouping

COUNT(*), SUM(col), MIN(col), MAX(col), AVG(col)

Definition: Aggregate functions; ignore NULLs.

GROUP BY

Definition: Groups rows for aggregates. Example: GROUP BY col;

HAVING

Definition: Filters groups after aggregation.
Example: HAVING AVG(x)>10;

ORDER BY

Definition: Sorts result set (ASC/DESC). Example: ORDER BY cnt DESC;

Eval Order

FROM → WHERE → GROUP BY → HAVING → SELECT → ORDER BY

Data Modification

-- update specific data with the WHERE clause

UPDATE table1 SET col1 = 1 WHERE col2 = 2

-- insert values manually

INSERT INTO table1 (ID, FIRST_NAME, LAST_NAME)
VALUES (1, 'Rebel', 'Labs');

-- or by using the results of a query

INSERT INTO table1 (ID, FIRST_NAME, LAST_NAME)
SELECT id, last_name, first_name FROM table2

Nested Queries

Correlated Subquery

Definition: Subquery evaluated per outer row.
Example:

```
SELECT a.actorname,
       (SELECT m.genre
        FROM Movie m
        WHERE m.name = a.moviename)
AS genre
FROM ActedIn a;
```

Subquery in FROM

Definition: Treat subquery as table. Example:

```
SELECT x.name, x.rating
FROM (SELECT * FROM Movie WHERE
      rating>8) AS x
WHERE x.rating<9;
```

EXISTS / IN

Definition: Existential tests in WHERE. Example:

```
WHERE EXISTS (
  SELECT 1 FROM Movie m
  WHERE m.name=a.moviename
  AND m.genre='Sci-Fi');
-- OR --
WHERE a.moviename IN (
  SELECT m.name FROM Movie m
  WHERE m.genre='Sci-Fi');
```

ALL / NOT EXISTS

Definition: Universal tests. Example:

```
WHERE 100000<ALL(
  SELECT a.salary FROM ActedIn a
  WHERE a.moviename=m.name);
-- OR --
WHERE NOT EXISTS (
  SELECT 1 FROM ActedIn a
  WHERE a.moviename=m.name
  AND a.salary<=100000);
```

Creating Tables

Customers / Products / Junction

Definition: Example of CREATE TABLE with PKs and FKs. Example:

```
CREATE TABLE Customers (
  CustomerID INT NOT NULL,
  FirstName VARCHAR(50),
  LastName VARCHAR(50),
  Email VARCHAR(100) UNIQUE,
  PRIMARY KEY (CustomerID)
);
CREATE TABLE Products (
  ProductID INT NOT NULL,
  ProductName VARCHAR(100),
  Price DECIMAL(10,2) DEFAULT 0.00,
  PRIMARY KEY (ProductID)
);
CREATE TABLE CustomerOrders (
  CustomerID INT NOT NULL,
  OrderID INT NOT NULL,
  OrderDate DATE NOT NULL,
  PRIMARY KEY (CustomerID, OrderID),
  FOREIGN KEY (CustomerID) REFERENCES
    Customers(CustomerID),
  FOREIGN KEY (OrderID) REFERENCES
    Orders(OrderID)
);
```

```
DROP TABLE IF EXISTS WorksOn;
DROP TABLE IF EXISTS Project;
DROP TABLE IF EXISTS Developer;
```

```
CREATE TABLE Project(
  pid INT PRIMARY KEY,
  name TEXT,
  startYear INT
);
```

```
CREATE TABLE Developer(
  did INT PRIMARY KEY,
  name TEXT,
  hireYear INT
);
```

```
CREATE TABLE WorksOn(
  pid INT REFERENCES Project(pid),
  did INT REFERENCES Developer(did),
  year INT
);
```

1. **Dev count per project-year** Returns for each project and for each year the total number of developers who worked on that project in that year. Your query should return the project's ID, name, year, and number of developers, sorted by year in increasing order, and, within each year, sorted in decreasing order of the number of developers.

TEXT OPERATORS
Fetch names of cities that start with a 'P' or end with an 'S':
SELECT name
FROM city
WHERE name LIKE 'P%' OR name LIKE '%S';

```
SELECT P.pid, P.name, W.year, COUNT(*)
      AS cnt
FROM Project AS P
JOIN WorksOn AS W ON P.pid = W.pid
GROUP BY P.pid, P.name, W.year
ORDER BY W.year ASC, cnt DESC;
```

2. **Inactive projects** A project is called inactive if no developer worked on it since 2010 (including and after 2010). Write a SQL query to find all inactive projects. You should return their IDs and names.

```
-- Solution 1
SELECT P.pid, P.name
FROM Project AS P
WHERE NOT EXISTS (
  SELECT 1
  FROM WorksOn AS W
  WHERE W.pid = P.pid AND W.year >= 2010
);
```

```
-- Solution 2
SELECT P.pid, P.name
FROM Project AS P
LEFT OUTER JOIN WorksOn AS W
ON P.pid = W.pid AND W.year >= 2010
GROUP BY P.pid, P.name
HAVING COUNT(W.pid) = 0;
```

3. **Top projects since 1990** For each year since 1990 (including and after 1990), return the project(s) that the most developers worked on during that year. Return the year and project ID. In case of a tie (if multiple projects had the maximum number of developers) return all of them.

```
-- Universal Quantifier Approach
SELECT W.year, W.pid
FROM WorksOn AS W
WHERE W.year >= 1990
GROUP BY W.year, W.pid
HAVING COUNT() >= ALL (
  SELECT COUNT()
  FROM WorksOn AS W2
  WHERE W2.year = W.year
GROUP BY W2.pid
);
```

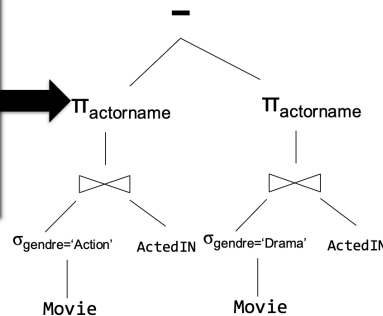
```
-- Witness Approach
WITH A AS (
  SELECT year, pid, COUNT(*) AS cnt
  FROM WorksOn
  WHERE year >= 1990
  GROUP BY year, pid
),
B AS (
```

```
SELECT year, MAX(cnt) AS maxcnt
FROM A
GROUP BY year
)
SELECT A.year, A.pid
FROM A JOIN B ON A.year = B.year AND
      A.cnt = B.maxcnt;
```

4. **SystemX devs** SystemX is the oldest project of the company. Write a SQL query that returns all developers who worked every year on SystemX, from when it started until 2015. Your query should return the developers' ID and name.

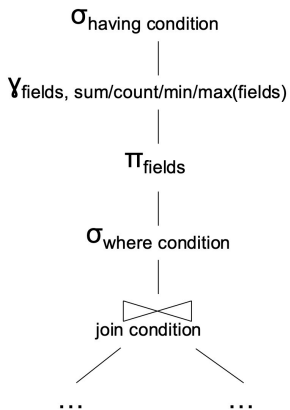
```
-- Solution 1
SELECT D.did, D.name
FROM Project AS P
JOIN WorksOn AS W ON P.pid = W.pid
JOIN Developer AS D ON W.did = D.did
WHERE P.name = 'SystemX'
AND W.year BETWEEN P.startYear AND 2015
GROUP BY D.did, D.name, P.startYear
HAVING COUNT(*) = (2015 - P.startYear + 1);
```

```
SELECT a1.actorname
FROM Movie m1, ActedIN a1
WHERE m1.name= a1.moviename
AND m1.genre='Action'
EXCEPT
SELECT a2.actorname
FROM Movie m2, ActedIN a2
WHERE m2.name= a2.moviename
AND m2.genre='Drama'
```



```
SELECT X.did
FROM Developer X
WHERE NOT EXISTS
```

```
(SELECT *
FROM Project Z
WHERE NOT EXISTS
  (SELECT *
   FROM WorksOn Y
   WHERE X.did = Y.did AND Y.pid = Z.pid
   AND Y.year = 2015));
```



SELECT fields
FROM R, S, ...
WHERE condition
GROUP BY fields
HAVING condition

```
SELECT X.did, X.name, count(*)
FROM Developer X, WorksOn y, Project Z
WHERE X.did = y.did and y.pid = Z.pid
AND y.year < Z.startYear + 2
GROUP BY X.did, X.name, X.hireYear
HAVING X.hireYear + 10 < max(startYear);
```

Solution:

$$\pi_{(X3.did)}(\rho_{X3}(Developer)) - \pi_{(X.did)}(A - B)$$

Where A is

$$\pi_{(Y2.pid, Y2.did)}(\rho_{Y2}(WorksOn))$$

Where B is

$$\pi_{(Z.pid, X.did)}(\sigma_{(Y.year=2015)}(\rho_X(Developer) \bowtie_{(X.did=Y.did)} \rho_Y(WorksOn) \bowtie_{(Y.pid=Z.pid)} \rho_Z(Project)))$$

Solution:

$$\pi_{(X.did, X.name, C)}(\sigma_{(X.hireYear+10 < M)}(\gamma_{(X.did, X.name, X.hireYear, count(*) \rightarrow C, \max(Z.startYear) \rightarrow M)}(\sigma_{(Y.year < Z.startYear+2)}(\rho_X(Developer) \bowtie_{(X.did=Y.did)} \rho_Y(WorkOn) \bowtie_{(Y.pid=Z.pid)} \rho_Z(Project)))))$$