Relational Algebra

SQL

Relational Query Languages

- Languages for describing queries on a relational database
- *Structured Query Language* (SQL)
  - Predominant application-level query language
  - Declarative
- *Relational Algebra*
  - Intermediate language used within DBMS
  - Procedural

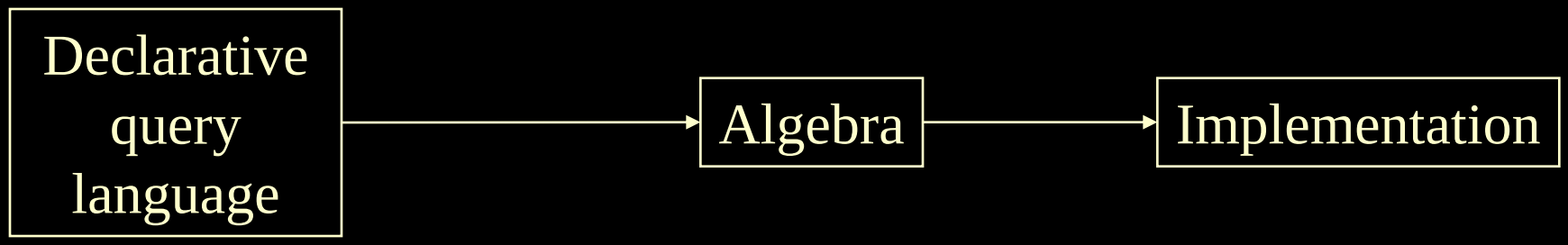
SQL

SQL consists of the following parts:

- Data Definition Language (DDL)
- Interactive Data Manipulation Language (Interactive DML)
- Embedded Data Manipulation Language (Embedded DML)
- Views
- Integrity
- Transaction Control
- Authorization
- Catalog and Dictionary Facilities

Relational Algebra

- Formalism for creating new relations from existing ones
- Its place in the big picture:



SQL
Relational Calculus

Relational Algebra

Relational Algebra

- Five operators:
  - Union: $\cup$
  - Difference: -
  - Selection: σ
  - Projection: Π
  - Cartesian Product: $\times$
- Derived or auxiliary operators:
  - Intersection, complement
  - Joins (natural, equi-join, etc.)
  - Renaming: ρ

Renaming

- Changes the schema, not the instance
- Notation: $\rho_{B1,\ldots,Bn}(R)$
- Example:
  - $\rho_{LastName,\ SocSocNo}$ (Employee)
  - Output schema:
    (LastName, SocSocNo)

Renaming Example

Employee

Name	SSN
John	999999999
Tony	777777777

$\rho_{LastName,\ SocSocNo}$ **(Employee)**

LastName	SocSocNo
John	999999999
Tony	777777777

Set Operators

- SQL provides UNION, EXCEPT (set difference), and INTERSECT for union compatible tables
- Example: Find all professors in the IT Department and all professors that have taught IT courses

```
(SELECT   P.Name
FROM   Professor P, Teaching T
WHERE  P.Id=T.ProfId AND T.CrsCode LIKE 'IT%')
UNION
(SELECT  P.Name
FROM  Professor P
WHERE  P.DeptId = 'IT')
```

Division

- *Strategy for implementing division in SQL*:
  - Find set, A, of all departments in which a particular professor, *p*, has taught a course
  - Find set, B, of all departments
  - Output *p* if A $\supseteq$ B, or, equivalently, if B–A is empty

Division – SQL Solution

```
SELECT P.Id
FROM Professor P
WHERE NOT EXISTS
    (SELECT D.DeptId          -- set B of all dept Ids
     FROM Department D
          EXCEPT
     SELECT C.DeptId          -- set A of dept Ids of depts in
                              -- which P has taught a course
     FROM Teaching T, Course C
     WHERE T.ProfId=P.Id      -- global variable
           AND T.CrsCode=C.CrsCode)
```

Aggregates

- Functions that operate on sets:
  - COUNT, SUM, AVG, MAX, MIN
- Produce numbers (not tables)
- Not part of relational algebra

```
SELECT COUNT(*)
FROM   Professor P
```

```
SELECT MAX (Salary)
FROM  Employee E
```

Aggregates

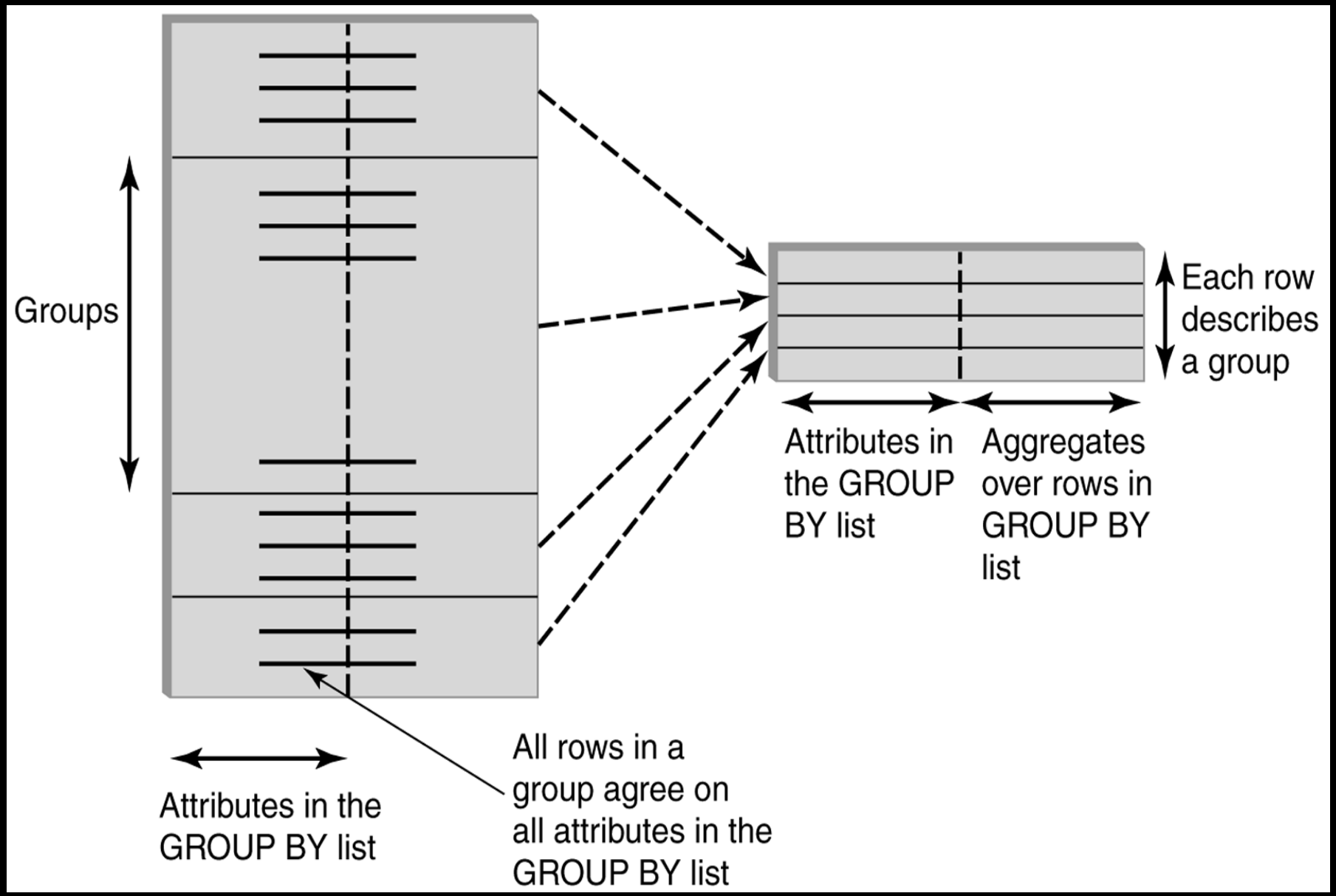
Count the number of courses taught in S2000

```
SELECT COUNT (T.CrsCode)
FROM Teaching T
WHERE  T.Semester = 'S2000'
```

But if multiple sections of same course are taught, use:

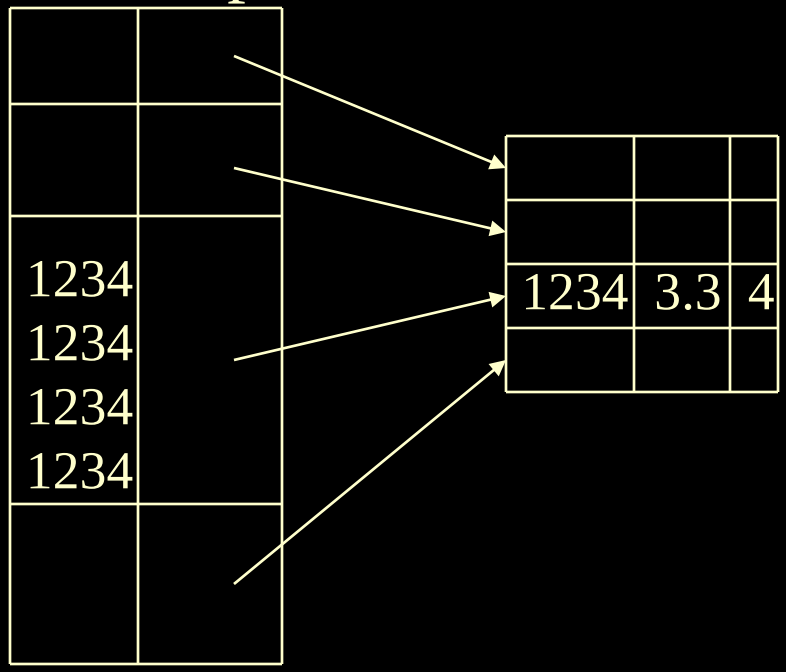
```
SELECT COUNT (DISTINCT T.CrsCode)
FROM Teaching T
WHERE  T.Semester = 'S2000'
```

GROUP BY


Groups
Each row describes a group
Attributes in the GROUP BY list
Aggregates over rows in GROUP BY list
Attributes in the GROUP BY list
All rows in a group agree on all attributes in the GROUP BY list

GROUP BY - *Example*

Transcript



Attributes:

-student's *Id*

-avg grade

-number of courses

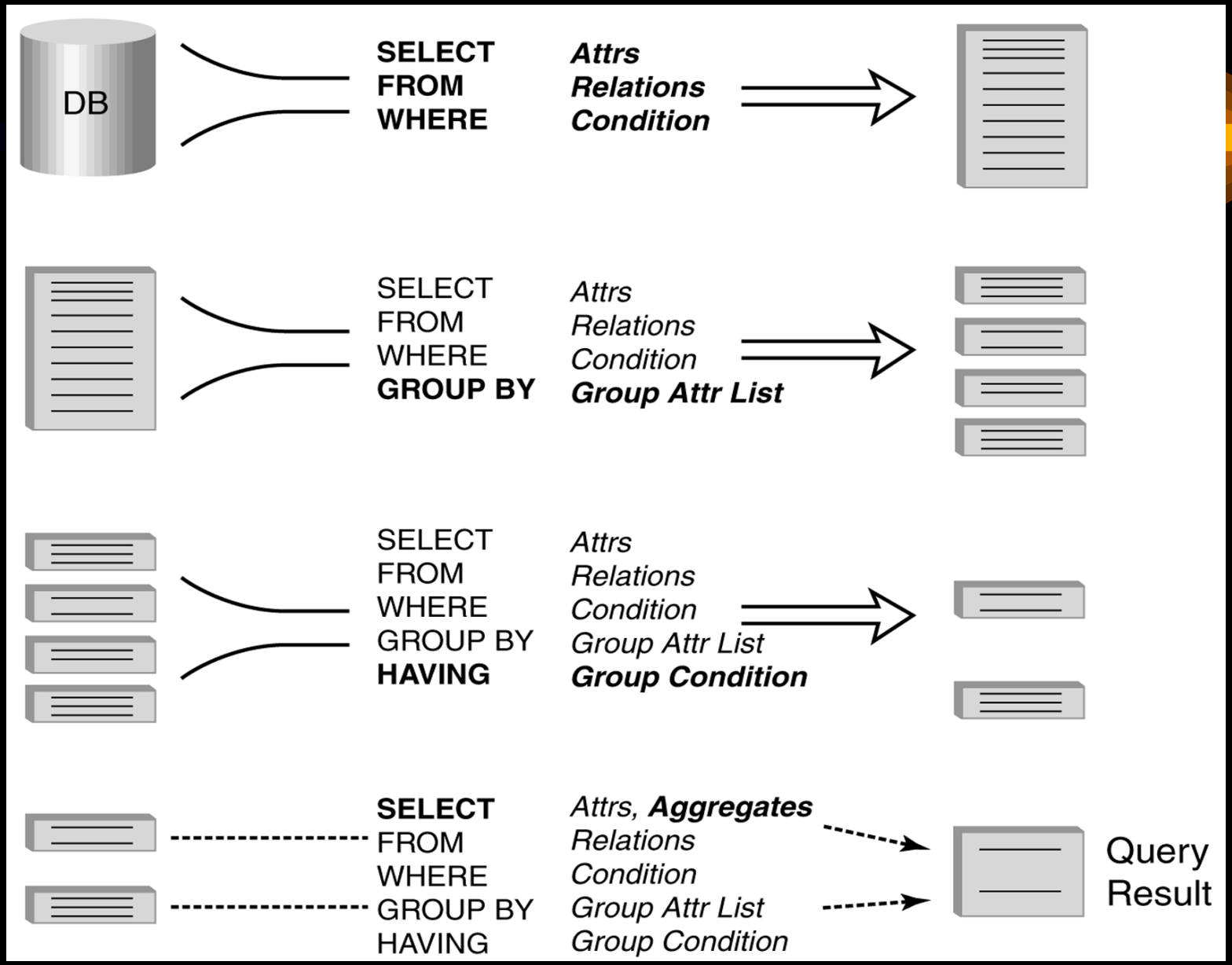
SELECT T.*StudId*, AVG(T.*Grade*), COUNT (*)
FROM Transcript T
GROUP BY T.*StudId*

HAVING Clause

- Eliminates unwanted groups (analogous to WHERE clause)
- HAVING condition constructed from attributes of GROUP BY list and aggregates of attributes not in list

```
SELECT  T.StudId, AVG(T.Grade) AS CumGpa,
                  COUNT (*) AS NumCrs
FROM    Transcript T
WHERE   T.CrsCode  LIKE  'CS%'
GROUP BY  T.StudId
HAVING  AVG (T.Grade) > 3.5
```

Evaluation of GroupBy with Having



Example

- Output the name and address of all seniors on the Dean's List

```
SELECT  S.Name,  S.Address
FROM  Student S, Transcript T
WHERE  S.StudId = T.StudId AND S.Status = 'senior'

GROUP BY  S.StudId                -- wrong
          S.Name, S.Address       -- right

HAVING AVG (T.Grade) > 3.5  AND  SUM (T.Credit) > 90
```

ORDER BY Clause

- Causes rows to be output in a specified order

```
SELECT  T.StudId, COUNT (*) AS NumCrs,
        AVG(T.Grade) AS CumGpa
FROM    Transcript T
WHERE   T.CrsCode LIKE 'CS%'
GROUP BY  T.StudId
HAVING  AVG (T.Grade) > 3.5
ORDER BY  DESC  CumGpa,  ASC StudId
```

Views

- Used as a relation, but rows are not physically stored.
  - The contents of a view is *computed* when it is used within an SQL statement
- View is the result of a SELECT statement over other views and base relations
- When used in an SQL statement, the view definition is substituted for the view name in the statement
  - SELECT statement can be nested in FROM clause

View - Example

```
CREATE VIEW  CumGpa (StudId, Cum) AS
  SELECT  T.StudId,  AVG (T.Grade)
  FROM Transcript T
  GROUP BY T.StudId

SELECT  S.Name, C.Cum
FROM  CumGpa C,  Student S
WHERE  C.StudId = S.StudId AND C.Cum > 3.5
```

View Benefits

- *Access Control*: Users not granted access to base tables. Instead they are granted access to the view of the database appropriate to their needs.
  - *External schema* is composed of views.
  - View allows owner to provide SELECT access to a subset of columns (analogous to providing UPDATE and INSERT access to a subset of columns)

Modifying Data - Update

```
UPDATE  Employee E
SET     E.Salary = E.Salary * 1.05
WHERE   E.Department = 'research'
```

- Updates rows in a single table
- All rows satisfying WHERE clause (general form, including subqueries, allowed) are updated

Updating Views

- Question: Since views look like tables to users, can they be updated?
- Answer: Yes – a view update changes the underlying base table to produce the requested change to the view

```
CREATE VIEW   CsReg (StudId, CrsCode, Semester) AS
SELECT    T.StudId, T. CrsCode, T.Semester
FROM      Transcript T
WHERE  T.CrsCode LIKE 'CS%' AND T.Semester='S2000'
```

Updating Views - Problem 1

INSERT INTO CsReg (*StudId*, *CrsCode*, *Semester*)
VALUES (1111, 'CSE305', 'S2000')

- **Question**: What value should be placed in attributes of underlying table that have been projected out (e.g., *Grade*)?
- **Answer**: NULL (assuming null allowed in the missing attribute) or DEFAULT

Updating Views - Problem 2

INSERT INTO **CsReg** (*StudId*, *CrsCode*, *Semester*)
VALUES (1111, 'ECO105', 'S2000')

- **Problem**: New tuple not in view
- **Solution**: Allow insertion (assuming the WITH CHECK OPTION clause has not been appended to the CREATE VIEW statement)

Updating Views - Problem 3

- Update to the view might not *uniquely* specify the change to the base table(s) that results in the desired modification of the view

```
CREATE VIEW ProfDept (PrName, DeName) AS
SELECT  P.Name, D.Name
FROM  Professor P, Department D
WHERE  P.DeptId = D.DeptId
```

Updating Views - Problem 3 (con't)

- Tuple <Smith, CS> can be deleted from ProfDept by:
  - Deleting row for Smith from Professor (but this is inappropriate if he is still at the University)
  - Deleting row for CS from Department (not what is intended)
  - Updating row for Smith in Professor by setting *DeptId* to null (seems like a good idea)

Updating Views - Restrictions

- Updatable views are restricted to those in which
  - No Cartesian product in FROM clause
  - no aggregates, GROUP BY, HAVING

For example, if we allowed:

```
CREATE VIEW AvgSalary (DeptId, Avg_Sal ) AS
    SELECT E.DeptId, AVG(E.Salary)
    FROM Employee E
    GROUP BY E.DeptId
```

then how do we handle:

```
UPDATE AvgSalary
    SET Avg_Sal = 1.1 * Avg_Sal
```

SQL: Join operation

- A join can be specified in the FROM clause which list the two input relations and the WHERE clause which lists the join condition.

Emp		Dept	
ID	State	ID	Division
1000	CA	1001	IT
1001	MA	1002	Sales
1002	TN	1003	Biotech

SQL: Join operation (cont.)

- inner join = join

  SELECT *

  FROM emp join dept (or FROM emp, dept)

  on emp.id = dept.id;

Emp.ID	Emp.State	Dept.ID	Dept.Division
1001	MA	1001	IT
1002	TN	1002	Sales

SQL: Join operation (cont.)

- left outer join = left join

```
SELECT *
FROM emp left join dept
on emp.id = dept.id;
```

Emp.ID	Emp.State	Dept.ID	Dept.Division
1000	CA	null	null
1001	MA	1001	IT
1002	TN	1002	Sales

SQL: Join operation (cont.)

- right outer join = right join

```
SELECT *
FROM emp right join dept
on emp.id = dept.id;
```

Emp.ID	Emp.State	Dept.ID	Dept.Division
1001	MA	1001	IT
1002	TN	1002	Sales
null	null	1003	Biotech

SQL: Like operation

Pattern matching selection

- % (arbitrary string)

  SELECT *

  FROM emp

  WHERE ID like ‘%01’;

  finds ID that ends with 01, e.g. 10<u>01</u>, 20<u>01</u>, etc

- _ (a single character)

  SELECT *

  FROM emp

  WHERE ID like ‘_01_’;

  finds ID that has the second and third character as 01, e.g. 1<u>01</u>0, 1<u>01</u>1, 1<u>01</u>2, 1<u>01</u>3, etc