SQL
Structured Query Language

Dr. Balasundaram A.
SCOPE, VIT Chennai

Overview

1. Introduction
2. SQL Data Types
3. DDL Commands
4. Integrity Constraints
5. DML Commands
6. Select Clauses
7. SQL Operators
8. SQL Joins
9. DCL Commands

1. Introduction

Some Facts on SQL

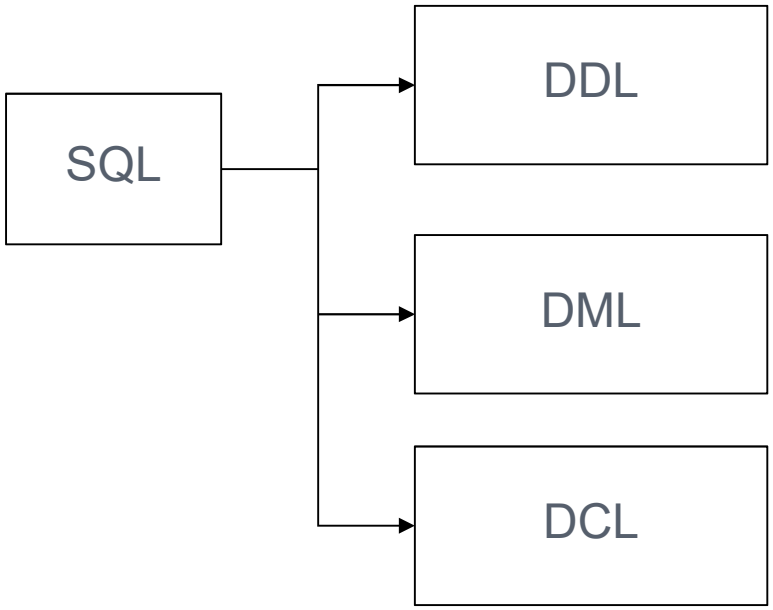
- SQL data is case-sensitive, SQL commands are not.
- First Version was developed at IBM by Donald D. Chamberlin and Raymond F. Boyce. [SQL]
- Developed using Dr. E.F. Codd's paper, "A Relational Model of Data for Large Shared Data Banks."
- Originally called SEQUEL from Structured English QUEry Language

NON-PROCEDURAL / PROCEDURAL

- SQL: language to access and manipulate data
- PL/SQL: a procedural extension to SQL language

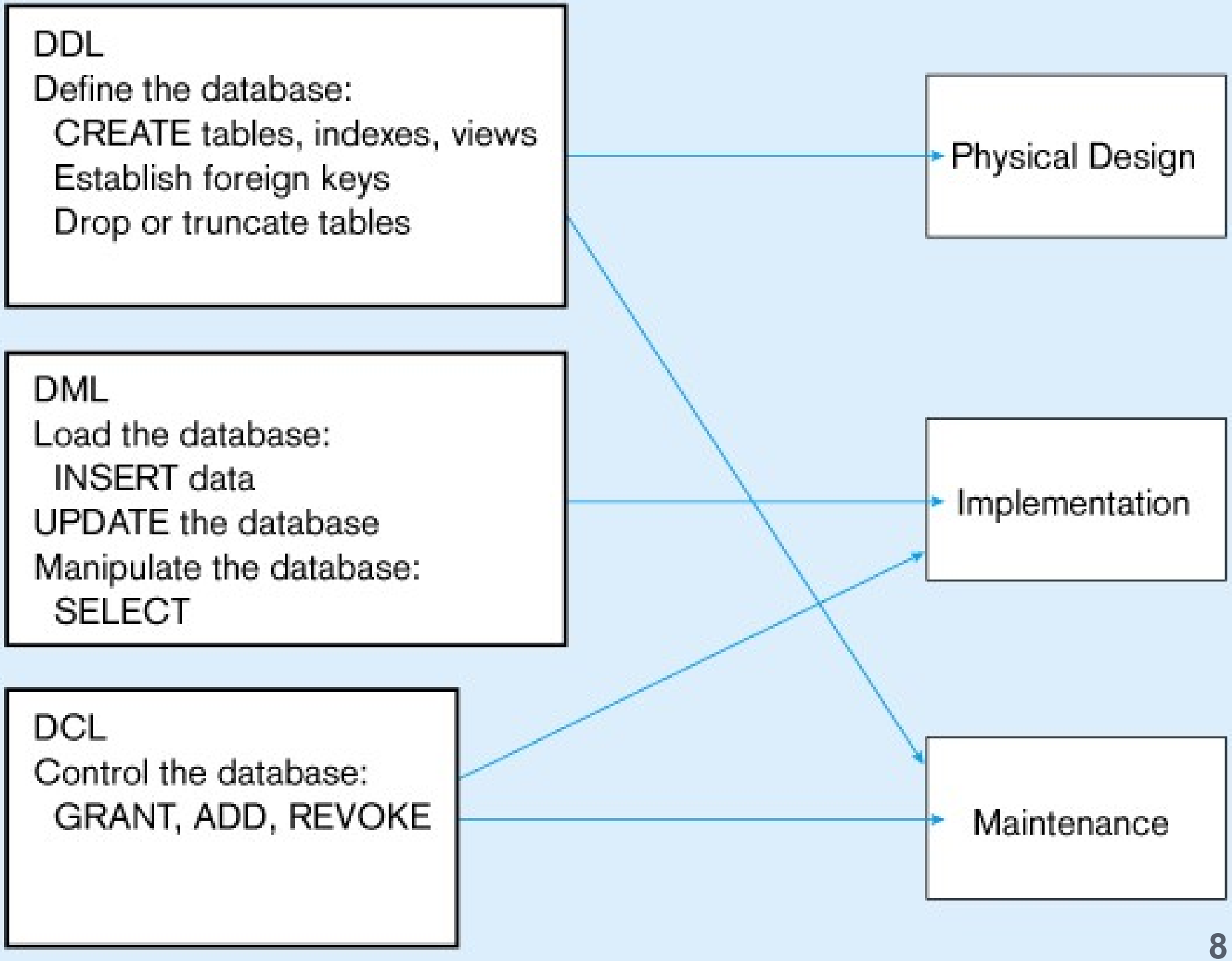
SQL

- Command line tool that process user's SQL statements
- Requires Oracle account



INTRODUCTION TO SQL

- SQL functions fit into three broad categories:
  - Data Definition Language (DDL)
    - statements that specify and modify database schemas.
    - SQL includes commands to:
      - Create database objects, such as tables, indexes, and views
      - Define access rights to those database objects
  - Data Manipulation Language (DML)
    - statements that manipulate database content.
    - Includes commands to insert, update, delete, and retrieve data within database tables
  - Data Control Language (DCL)
    - Commands that control a database, including administering privileges and committing data


DDL
Define the database:
CREATE tables, indexes, views
Establish foreign keys
Drop or truncate tables
DML
Load the database:
INSERT data
UPDATE the database
Manipulate the database:
SELECT
DCL
Control the database:
GRANT, ADD, REVOKE
Physical Design
Implementation
Maintenance

2. SQL Data Types

SQL Data types (from Oracle 9i)

- String types
  - CHAR(n) – fixed-length character data, n characters long Maximum length = 2000 bytes
  - VARCHAR2(n) – variable length character data, maximum 4000 bytes
  - LONG – variable-length character data, up to 4GB. Maximum 1 per table
- Numeric types
  - NUMBER(p,q) – general purpose numeric data type
  - INTEGER(p) – signed integer, p digits wide
  - FLOAT(p) – floating point in scientific notation with p binary digits precision
- Date/time type
  - DATE – fixed-length date/time in dd-mm-yy form

3. SQL: DDL Commands

- CREATE
- ALTER
- DROP

TABLE

SQL Data Definition Commands

COMMAND OR OPTION	DESCRIPTION
CREATE SCHEMA AUTHORIZATION	Creates a database schema
CREATE TABLE	Creates a new table in the user's database schema
NOT NULL	Ensures that a column will not have null values
UNIQUE	Ensures that a column will not have duplicate values
PRIMARY KEY	Defines a primary key for a table
FOREIGN KEY	Defines a foreign key for a table
DEFAULT	Defines a default value for a column (when no value is given)
CHECK	Constraint used to validate data in an attribute
CREATE INDEX	Creates an index for a table
CREATE VIEW	Creates a dynamic subset of rows/columns from one or more tables
ALTER TABLE	Modifies a table's definition (adds, modifies, or deletes attributes or constraints)
CREATE TABLE AS	Creates a new table based on a query in the user's database schema
DROP TABLE	Permanently deletes a table (and thus its data)
DROP INDEX	Permanently deletes an index
DROP VIEW	Permanently deletes a view

MAJOR CREATE STATEMENTS

- CREATE TABLE – defines a table and its columns
- CREATE SCHEMA – defines a portion of the database owned by a particular user
- CREATE VIEW – defines a logical table from one or more views

SQL: DDL Commands – Working with tables

- CREATE TABLE: used to create a table.
- ALTER TABLE: modifies a table after it was created.
- DROP TABLE: removes a table from a database.

SQL: CREATE TABLE Statement

- Things to consider before you create your table are:
  - the table name
  - the names of the columns
  - the type of data
  - what column(s) will make up the primary key
- CREATE TABLE statement syntax:

```
CREATE TABLE <table name>
( field1 datatype ( size ) constraints,
  field2 datatype ( size) constraints,
.....
);
```

Constraints are optional

SQL: ALTER TABLE Statement

- To add or drop columns on existing tables.
- ALTER TABLE statement syntax:

```
ALTER TABLE <table name>
ADD attr datatype;
Or
MODIFY old COLUMN attr new COLUMN  attr
;
Or
DROP COLUMN attr;
```

SQL: DROP TABLE Statement

Syntax

DROP TABLE statement syntax:
DROP TABLE <table name> [RESTRICT|CASCADE];

Two options:

- CASCADE: Specifies that any foreign key constraint violations that are caused by dropping the table will cause the corresponding rows of the related table to be deleted.

- RESTRICT: blocks the deletion of the table of any foreign key constraint violations would be created.

Example:

```
CREATE  TABLE FoodCart (
date varchar(10),
food varchar(20),
profit float
);
```

FoodCart

date	food	profit

```
ALTER  TABLE FoodCart (
ADD  sold int
);
```

FoodCart

date	food	profit	sold

```
ALTER  TABLE FoodCart(
DROP  COLUMN profit
);
```

FoodCart

date	food	sold

```
DROP  TABLE FoodCart;
```

RENAME Statement

With RENAME statement you can rename a table. Some of the relational database management system (RDBMS) does not support this command, because this is not standardizing statement.

RENAME TABLE {tbl_name} TO {new_tbl_name};

or

ALTER TABLE {tbl_name} RENAME TO {new_tbl_name};

4. Integrity Constraints

SQL Integrity Constraints

1. **Key Constraints**
- PRIMARY KEY Constraint
  - Ensures that all values in column are unique and NOT NULL
- UNIQUE KEY constraint
  - Ensures that all values in column are unique

2. **Attribute Constraints**
- NOT NULL constraint
  - Ensures that column does not accept nulls
- DEFAULT constraint
  - Assigns value to attribute when a new row is added to table
- CHECK constraint
  - Validates data when attribute value is entered

3. **Referential Integrity Constraints**
- FOREIGN KEY constraint
  - Defines a foreign key for a table

1. KEY CONSTRAINTS

Idea: specifies that a relation is a set, not a bag

SQL examples:

1. **Primary Key:**

```
CREATE TABLE branch(
        bname  CHAR(15)  PRIMARY KEY,
        bcity      CHAR(20),
        assets     INT);
```

or

```
CREATE TABLE depositor(
        cname   CHAR(15),
        acct_no  CHAR(5),
        PRIMARY KEY(cname, acct_no));
```

2. **Candidate/Unique Keys:**

```
CREATE TABLE customer (
        ssn     CHAR(9)    PRIMARY KEY,
        cname     CHAR(15),
        address   CHAR(30),
        city  CHAR(10),
        UNIQUE (cname, address, city);
```

KEY CONSTRAINTS

Effect of SQL Key declarations

PRIMARY (A1, A2, .., An) or
UNIQUE (A1, A2, ..., An)

Insertions: check if any tuple has same values for A1, A2, .., An as any inserted tuple. If found, **reject insertion**

Updates to any of A1, A2, ..., An: treat as insertion of entire tuple

Primary Vs Unique (candidate)

- 1 primary key per table, several unique keys allowed.
- Only primary key can be referenced by “foreign key” (ref integrity)
- DBMS may treat primary key differently
  (e.g.: implicitly create an index on PK)
- NULL values permitted in UNIQUE keys but not in PRIMARY KEY

2. Attribute Constraints

Idea:

- Attach constraints to values of attributes
- Enhances types system (e.g.: >= 0 rather than integer)

In SQL:

1. NOT NULL

```
e.g.:   CREATE TABLE branch(
                bname   CHAR(15)  NOT NULL,
                ....
                )
```

Note: declaring bname as primary key also prevents null values

2. **CHECK**

```
e.g.:   CREATE TABLE depositor(
                ....
                balance int NOT NULL,
                CHECK( balance >= 0),
                ....
                )
```

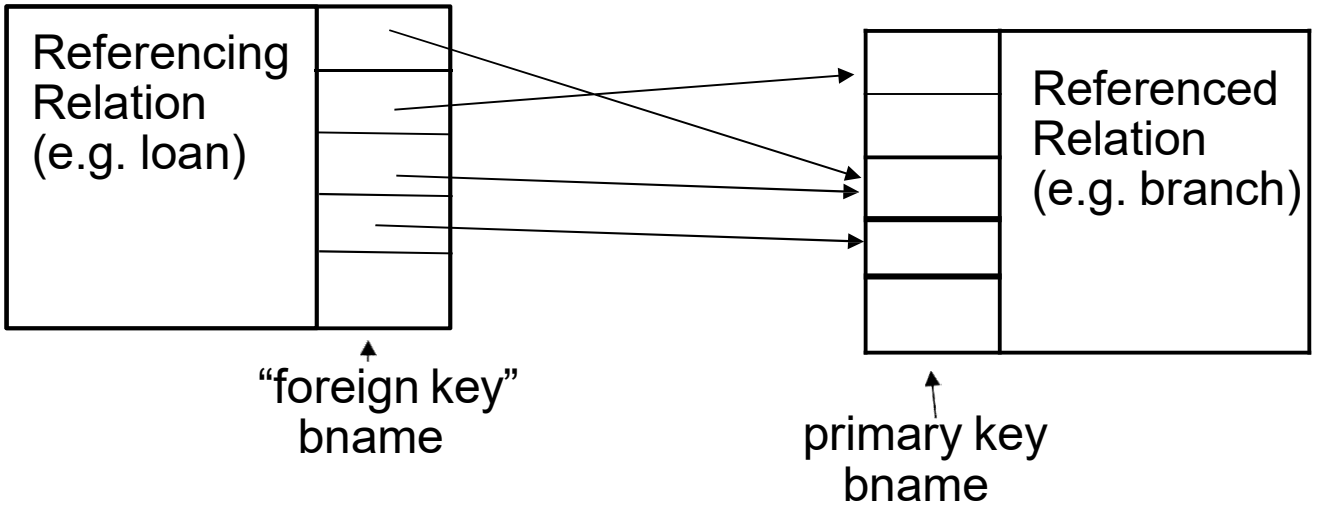
affect insertions, update in affected columns

CHECK CONSTRAINT - EG

```
CREATE TABLE credit_card(
                                ....
                                balance int NOT NULL,
                                 CHECK(  balance >= 0),
                                 CHECK (balance < limit),
                                 ....
                                 )
```

3. Referential Integrity Constraints

Idea: prevent "dangling tuples" (e.g.: a loan with a bname of 'Kenmore' when no Kenmore tuple is not in branch table)



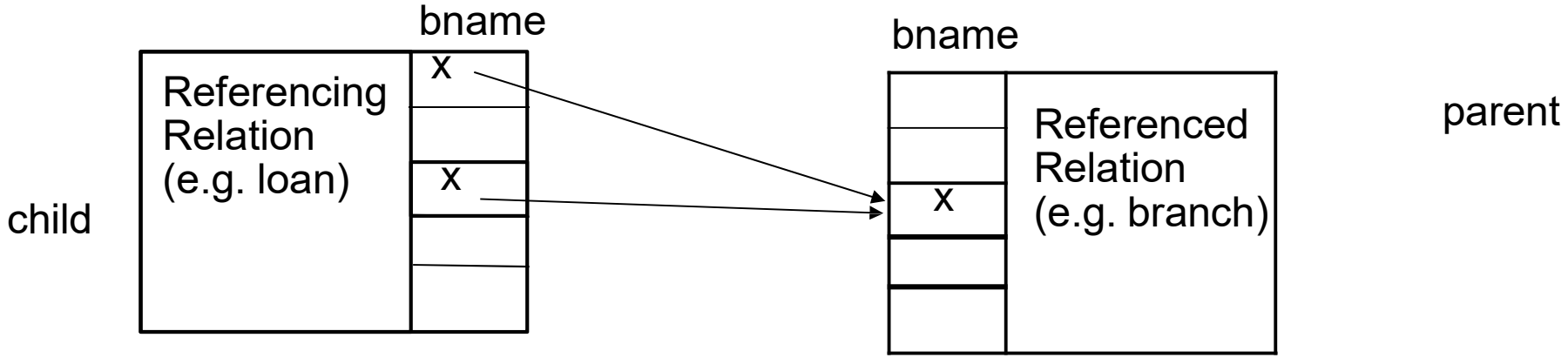
Ref Integrity:

ensure that:

foreign key value → primary key value

(note: need not to ensure ←, i.e., not all branches have to have loans)

REFERENTIAL INTEGRITY CONSTRAINTS



In SQL:

```
CREATE TABLE   branch(
        bname   CHAR(15)   PRIMARY KEY
        ....)

CREATE TABLE loan (
        .........
        FOREIGN KEY bname REFERENCES branch);
```

```
CREATE TABLE loan (
                .........
bname REFERENCES branch(bname));
```

Affects:

1) Insertions, updates of referencing relation
2) Deletions, updates of referenced relation

Referential Integrity in SQL

- When a referential-integrity constraint is violated, the normal procedure is to reject the action that caused the violation (that is, the transaction performing the update action is rolled back).

Cascade Update and Delete in SQL

create table *course*

(. . .

foreign key (*dept name) references department*

on delete cascade

on update cascade,

. . .);

5. DML Commands

- INSERT: adds new rows to a table.
- UPDATE: modifies one or more attributes.
- DELETE: deletes one or more rows from a table.
- SELECT: Display the contents of a table.

TABLE

SQL Data Manipulation Commands

COMMAND OR OPTION	DESCRIPTION
INSERT	Inserts row(s) into a table
SELECT	Selects attributes from rows in one or more tables or views
WHERE	Restricts the selection of rows based on a conditional expression
GROUP BY	Groups the selected rows based on one or more attributes
HAVING	Restricts the selection of grouped rows based on a condition
ORDER BY	Orders the selected rows based on one or more attributes
UPDATE	Modifies an attribute's values in one or more table's rows
DELETE	Deletes one or more rows from a table
COMMIT	Permanently saves data changes
ROLLBACK	Restores data to their original values

SQL: INSERT Statement

- To insert a row into a table, it is necessary to have a value for each attribute, and order matters.
- INSERT statement syntax:

```
INSERT INTO <table name>
VALUES ('value1', value2,....);
```

Example: INSERT INTO FoodCart
VALUES ('02/26/08','pizza',350);

FoodCart

date	food	sold
02/25/08	pizza	350
02/26/08	hotdog	500

date	food	sold
02/25/08	pizza	350
02/26/08	hotdog	500
02/26/08	pizza	70^{32}

INSERT INTO only the specific columns

```
INSERT
INTO table_name (column1,column2,column3,...)
VALUES (value1,value2,value3,...);
```

```
INSERT
INTO table_name VALUES('&column1',&column2,
&column3,...);
```

SQL: UPDATE Statement

- To update the content of the table:

UPDATE statement syntax:

```
UPDATE <table name> SET <attr> = <value>
WHERE <selection condition>;
```

Example: UPDATE FoodCart SET sold = 349
WHERE date = '02/25/08' AND food = 'pizza';

FoodCart

date	food	sold
02/25/08	pizza	350
02/26/08	hotdog	500
02/26/08	pizza	70

date	food	sold
02/25/08	pizza	349
02/26/08	hotdog	500
02/26/08	pizza	70_{34}

SQL: DELETE Statement

- To delete rows from the table:

DELETE statement syntax:

```
DELETE FROM <table name>
WHERE <condition>;
```

Example: DELETE FROM FoodCart
WHERE food = 'hotdog';

FoodCart

date	food	sold
02/25/08	pizza	349
02/26/08	hotdog	500
02/26/08	pizza	70

date	food	sold
02/25/08	pizza	349
02/26/08	pizza	70

Note: If the WHERE clause is omitted all rows of data are deleted from the table.

Basic SELECT Statement

- A basic SELECT statement includes 3 clauses

SELECT <attribute name> FROM <tables> WHERE <condition>

SELECT	FROM	WHERE
Specifies the attributes that are part of the resulting relation	Specifies the tables that serve as the input to the statement	Specifies the selection condition, including the join condition.

Note: that you don't need to use WHERE

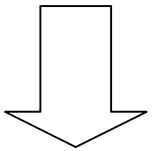
Simple SQL Query

Using a "*" in a select statement indicates that every attribute of the input table is to be selected.

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT  *
FROM    Product
WHERE   category='Gadgets'
```



"selection"

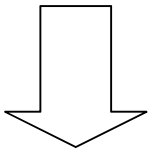
PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks

Simple SQL Query

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT   PName, Price, Manufacturer
FROM     Product
WHERE    Price > 100
```



"selection" and "projection"

PName	Price	Manufacturer
SingleTouch	$149.99	Canon
MultiTouch	$203.99	Hitachi

Distinct in SELECT Statement

- To get unique rows, type the keyword DISTINCT after SELECT.

Example: SELECT **DISTINCT** * FROM …WHERE …;

Example:

Person

Name	Age	Weight
Harry	34	80
Sally	28	64
George	29	70
Helena	54	54
Peter	34	80

1) SELECT *
FROM person
WHERE age > 30;

Name	Age	Weight
Harry	34	80
Helena	54	54
Peter	34	80

2) SELECT weight
FROM person
WHERE age > 30;

Weight
80
54
80

3) SELECT **distinct** weight
FROM person
WHERE age > 30;

Weight
80
54

SQL: Aggregate Functions

Are used to provide summarization information for SQL statements, which return a single value.

COUNT(attr)
SUM(attr)
MAX(attr)
MIN(attr)
AVG(attr)

Note: when using aggregate functions, NULL values are not considered, except in COUNT(*) .

SQL: Aggregate Functions (cont.)

FoodCart

date	food	sold
02/25/08	pizza	349
02/26/08	hotdog	500
02/26/08	pizza	70

COUNT(attr) -> return # of rows that are not null

Ex: COUNT(distinct food)from FoodCart; -> 2

SUM(attr) -> return the sum of values in the attr

Ex: SUM(sold) from FoodCart; -> 919

MAX(attr) -> return the highest value from the attr

Ex: MAX(sold) from FoodCart; -> 500

SQL: Aggregate Functions (cont.)

FoodCart

date	food	sold
02/25/08	pizza	349
02/26/08	hotdog	500
02/26/08	pizza	70

MIN(attr) -> return the lowest value from the attr
Ex: MIN(sold) from FoodCart; -> 70

AVG(attr) -> return the average value from the attr
Ex: AVG(sold) from FoodCart; -> 306.33
Note: value is rounded to the precision of the datatype

6. SELECT Clauses

- Group By
- Having
- Order By
- Like

- Clauses of the SELECT statement:
  - GROUP BY
    - Indicate **columns** to group the results
  - HAVING
    - Indicate the **conditions** under which a **group** will be included
  - ORDER BY
    - Sorts the result according to specified **columns**

The GROUP BY Clause

- The function to **divide the tuples into groups** and returns an aggregate for each group.
- Usually, it is an aggregate function's companion

```
SELECT food, sum(sold) as totalSold
FROM FoodCart
group by food;
```

FoodCart

date	food	sold
02/25/08	pizza	349
02/26/08	hotdog	500
02/26/08	pizza	70

food	totalSold
hotdog	500
pizza	419

The Situation: Student Particulars

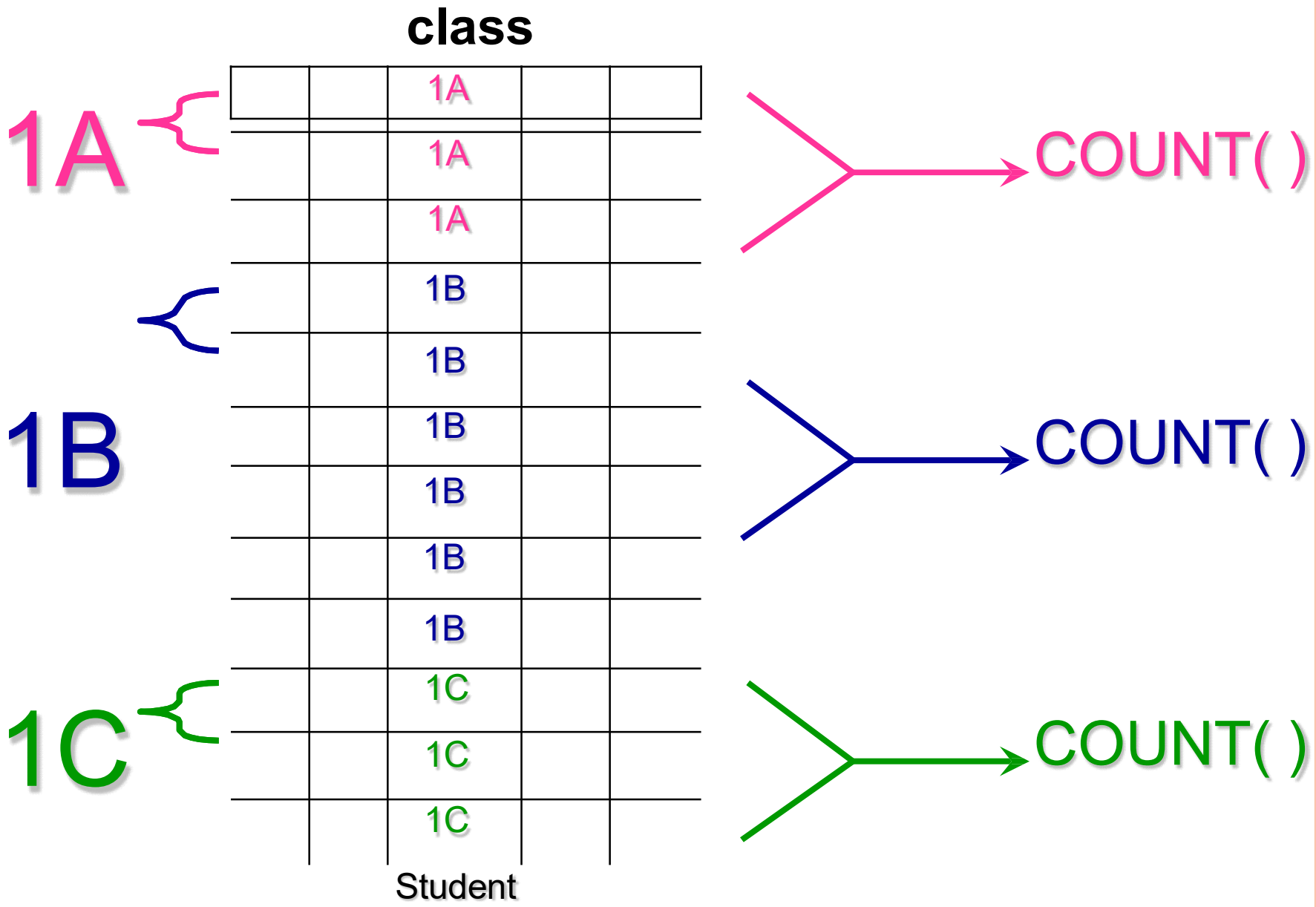
field	type	width	contents
id	numeric	4	student id number
name	character	10	name
dob	date	8	date of birth
sex	character	1	sex: M / F
class	character	2	class
hcode	character	1	house code: R, Y, B, G
dcode	character	3	district code
remission	logical	1	fee remission
mtest	numeric	2	Math test score

id	name	dob	sex	class	mtest	hcode	dcode	remission
9801	Peter	06/04/86	M	1A	70	R	SSP	.F.
9802	Mary	01/10/86	F	1A	92	Y	HHM	.F.
9803	Johnny	03/16/86	M	1A	91	G	SSP	.T.
9804	Wendy	07/09/86	F	1B	84	B	YMT	.F.
9805	Tobe	10/17/86	M	1B	88	R	YMT	.F.
:	:	:	:	:	:	:	:	:

Question:

List the number of students of each class.

⇩Group By Class



Group By Class

Eg: List the number of students of each class.

```
SELECT class, COUNT(*) FROM student
GROUP BY class;
```

Result

class	cnt
1A	10
1B	9
1C	9
2A	8
2B	8
2C	6

Group By Class

Eg: List the average Math test score of each class.

```
SELECT class, AVG(mtest) FROM student
GROUP BY class;
```

Result

class	avg_mtest
1A	85.90
1B	70.33
1C	37.89
2A	89.38
2B	53.13
2C	32.67

Group By Class

Eg: List the number of girls of each district.

```
SELECT dcode, COUNT(*) FROM student
WHERE sex="F" GROUP BY dcode;
```

Result

dcode	cnt
HHM	6
KWC	1
MKK	1
SSP	5
TST	4
YMT	8

The HAVING Clause

- The substitute of WHERE for aggregate functions
- Usually, it is **an aggregate function's companion**

AGGREGATE FUNCTIONS	Used with SELECT to return mathematical summaries on columns
COUNT	Returns the number of rows with non-null values for a given column
MIN	Returns the minimum attribute value found in a given column
MAX	Returns the maximum attribute value found in a given column
SUM	Returns the sum of all values for a given column
AVG	Returns the average of all values for a given column

The HAVING Clause – Example 1

```
SELECT food, sum(sold) as totalSold
FROM FoodCart
group by food
having sum(sold) > 450;
```

FoodCart

date	food	sold
02/25/08	pizza	349
02/26/08	hotdog	500
02/26/08	pizza	70

food	totalSold
hotdog	500

The HAVING Clause – Example 2

```
SELECT STATE, COUNT(STATE)
FROM CUSTOMER_V
GROUP BY STATE
HAVING COUNT(STATE) > 1;
```

GROUP BY AND HAVING

Eg: List the average Math test score of the boys in each class. The list should not contain class with less than 3 boys.

```
SELECT AVG(mtest), class FROM student
WHERE sex="M" GROUP BY class
HAVING COUNT(*) >= 3;
```

Result

avg_mtest	class
86.00	1A
77.75	1B
35.60	1C
86.50	2A
56.50	2B

The ORDER BY Clause

Ordered result selection

- desc (descending order)

```
SELECT *
FROM emp
order by state desc
```

→ puts state in descending order, e.g. TN, MA, CA

- asc (ascending order)

```
SELECT *
FROM emp
order by id asc
```

→ puts ID in ascending order, e.g. 1001, 1002, 1003

DISPLAY ORDER

Eg: List the boys of class 1A, order by their names.

```
SELECT name, id FROM student
WHERE sex="M" AND class="1A" ORDER BY
name;
```

name	id
Peter	9801
Johnny	9803
Luke	9810
Bobby	9811
Aaron	9812
Ron	9813

name	id
Aaron	9812
Bobby	9811
Johnny	9803
Luke	9810
Peter	9801
Ron	9813

DISPLAY ORDER

Eg: List the number of students of each district (in desc. order).

```
SELECT COUNT(*) AS cnt, dcode FROM
student GROUP BY dcode ORDER BY cnt DESC;
```

Result

cnt	docode
11	YMT
10	HHM
10	SSP
9	MKK
5	TST
2	TSW
1	KWC
1	MMK
1	SHT

Ordering the Results

Product

PName	Price	Category	Manufacturer
Gizmo	$19.99	Gadgets	GizmoWorks
Powergizmo	$29.99	Gadgets	GizmoWorks
SingleTouch	$149.99	Photography	Canon
MultiTouch	$203.99	Household	Hitachi

```
SELECT  category
FROM    Product
ORDER BY  pname
```

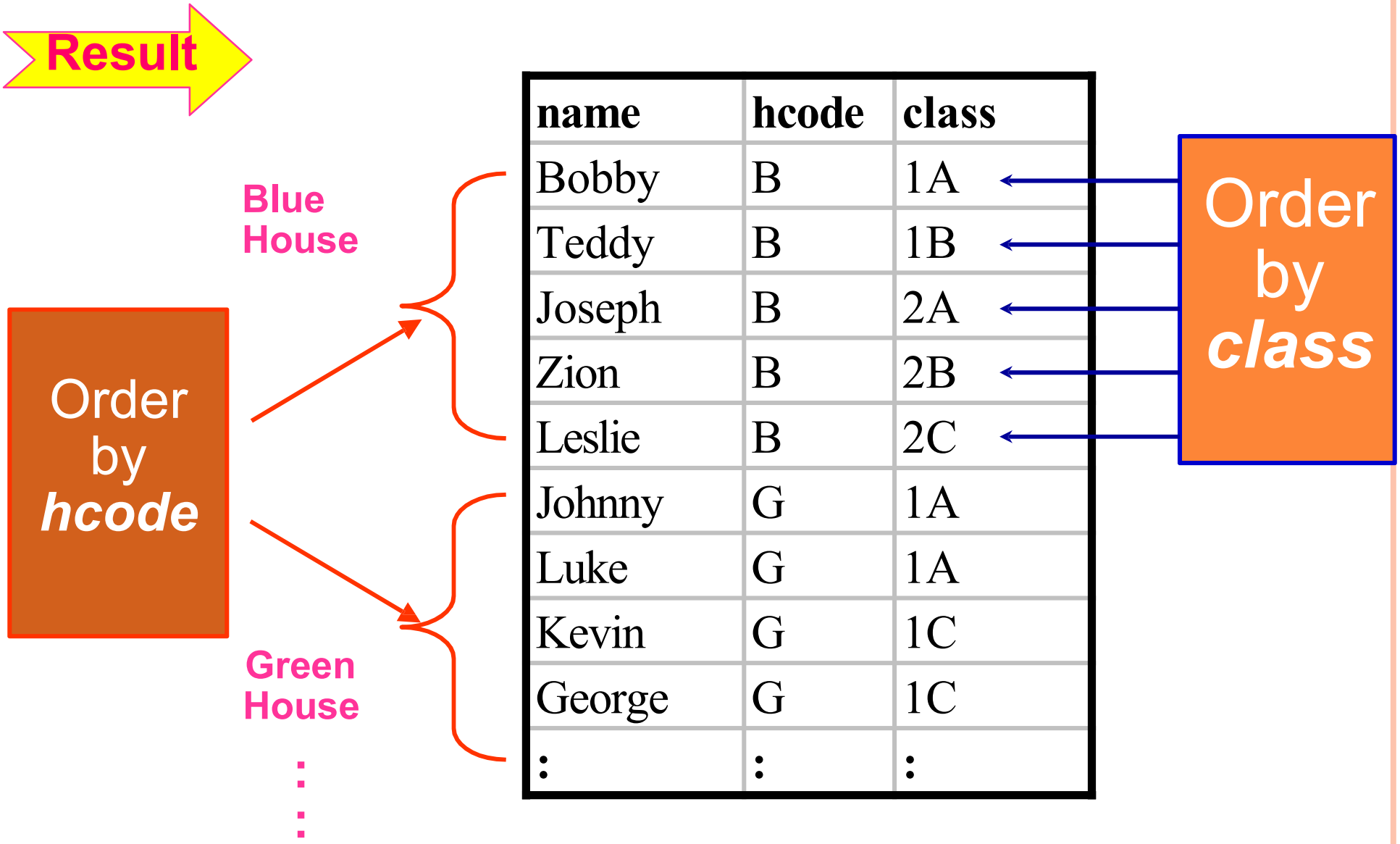
Category
Gadgets
Household
Gadgets
Photography

Display Order

Eg. List the boys of each house order by the classes. (2-level ordering)

```
SELECT name, class, hcode FROM student
WHERE sex="M" ORDER BY hcode, class;
```

DISPLAY ORDER



7. SQL OPERATORS

COMMAND OR OPTION	DESCRIPTION
COMPARISON OPERATORS	
=, <, >, <=, >=, <>	Used in conditional expressions
LOGICAL OPERATORS	
AND/OR/NOT	Used in conditional expressions
SPECIAL OPERATORS	Used in conditional expressions
BETWEEN	Checks whether an attribute value is within a range
IS NULL	Checks whether an attribute value is null
LIKE	Checks whether an attribute value matches a given string pattern
IN	Checks whether an attribute value matches any value within a value list
EXISTS	Checks whether a subquery returns any rows
DISTINCT	Limits values to unique values

BETWEEN

Eg: List the 1A students whose Math test score is between 80 and 90 (incl.)

```
SELECT name, mtest FROM student
WHERE class="1A" AND
mtest BETWEEN 80 AND 90;
```

Result

name	mtest
Luke	86
Aaron	83
Gigi	84

IN

List the students with their class who were born on Wednesdays or Saturdays.

```
SELECT name, class, CDOW(dob) AS bdate FROM student WHERE DOW(dob) IN (4,7)
```

Result

name	class	bdate
Peter	1A	Wednesday
Wendy	1B	Wednesday
Kevin	1C	Saturday
Luke	1A	Wednesday
Aaron	1A	Saturday
:	:	:

CDOW( ) returns the day
DOW() returns the day number - numeric day-of-the-week value

NOT IN

List the students who were not born in January, March, June, September.

```
SELECT name, class, dob FROM student
WHERE MONTH(dob) NOT IN (1,3,6,9);
```

Result

name	class	dob
Wendy	1B	07/09/86
Tobe	1B	10/17/86
Eric	1C	05/05/87
Patty	1C	08/13/87
Kevin	1C	11/21/87
Bobby	1A	02/16/86
Aaron	1A	08/02/86
:	:	:

THE **LIKE** OPERATOR

- **LIKE** : pattern matching on strings
- NOT LIKE – Not in the pattern string
- p may contain two special symbols:
  - % = any sequence of characters
  - _ = any single character

Find all products whose name mentions ‘gizmo’:

```
SELECT   *
FROM  Products
WHERE   PName LIKE ‘%gizmo%’
```

SQL: LIKE OPERATION

```
SELECT *
FROM emp
WHERE ID like '%01';
```

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

```
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

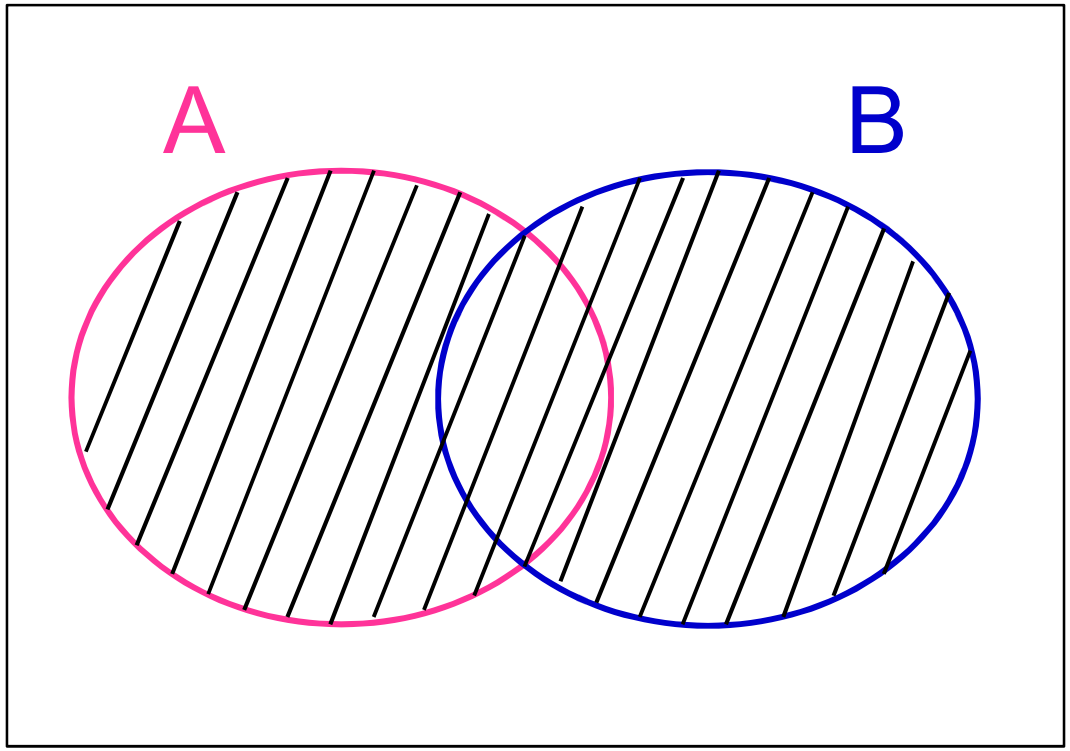
ENABLE AND DISABLE CONSTRAINTS

```
-- Disable all table constraints
ALTER TABLE YourTableName NOCHECK
CONSTRAINT ALL
-- Enable all table constraints
ALTER TABLE YourTableName CHECK
CONSTRAINT ALL
-- ---------
-- Disable single constraint
ALTER TABLE YourTableName NOCHECK
CONSTRAINT YourConstraint
-- Enable single constraint
ALTER TABLE YourTableName CHECK
CONSTRAINT YourConstraint
```

Set Operators

- Union
- Intersection
- Minus

union



A table containing all the rows from **A** and **B**.

UNION

```
SELECT ..... FROM ..... WHERE ..... ;
UNION
SELECT ..... FROM ..... WHERE .....;
```

eg. The two clubs want to hold a joint party. Make a list of all students. (Union)

Result

```
SELECT * FROM bridge
UNION
SELECT * FROM chess
ORDER BY class, name INTO TABLE party
```

- SELECT supplier_id FROM suppliers

UNION

SELECT supplier_id FROM orders ORDER BY supplier_id;

SELECT supplier_id, supplier_name FROM suppliers WHERE supplier_id > 2000

UNION

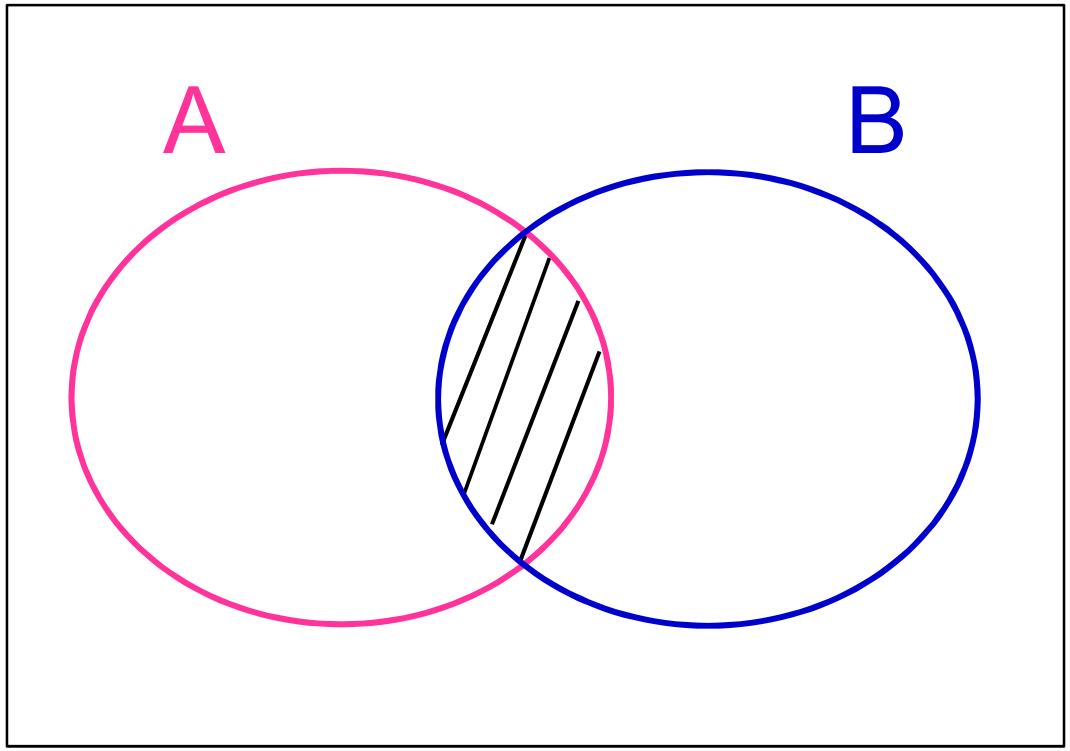
SELECT company_id, company_name FROM companies WHERE company_id > 1000 ORDER BY 1;

supplier_id	supplier_name
1000	Microsoft
2000	Oracle
3000	Apple
4000	Samsung

company_id	company_name
1000	Microsoft
3000	Apple
7000	Sony
8000	IBM

ID_Value	Name_Value
3000	Apple
4000	Samsung
7000	Sony
8000	IBM

intersection



A table containing only rows that appear in both **A** and **B**.

INTERSECTION

```
SELECT ..... FROM table1 ;
WHERE col IN  ( SELECT col FROM table2 )
```

eg. list the students who are members of both clubs. (Intersection)

Result

```
SELECT * FROM bridge
WHERE id IN ( SELECT id FROM chess ) ;
```

INTERSECT CLAUSE

- SELECT column1 [, column2] FROM tables [WHERE condition]

  INTERSECT

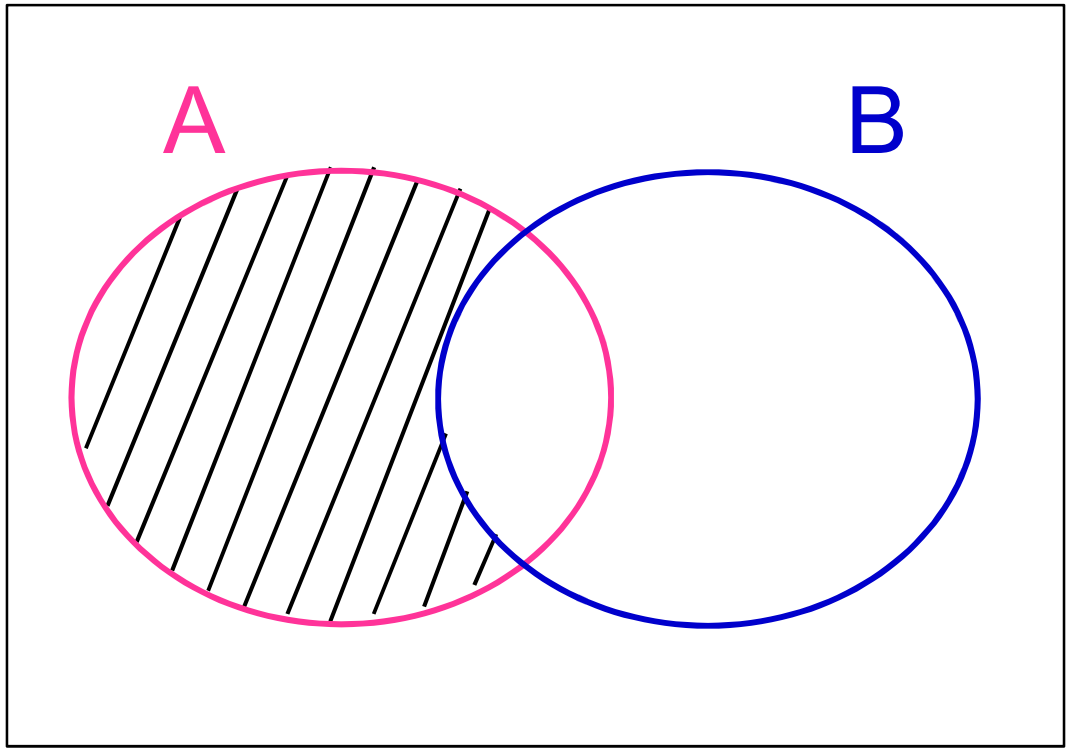
  SELECT column1 [, column2] FROM tables [WHERE condition]

INTERSECT - EXAMPLE

- SELECT supplier_id FROM suppliers
  INTERSECT
  SELECT supplier_id FROM orders;
- SELECT supplier_id FROM suppliers WHERE supplier_id > 78
  INTERSECT
  SELECT supplier_id FROM orders WHERE quantity <> 0;

MINUS

difference



A table containing rows that appear in **A** but not in **B**.

MINUS

```
SELECT ..... FROM table1 ;
WHERE col NOT IN  ( SELECT col FROM table2 )
```

eg. list the students who are members of the Bridge Club but not Chess Club. (Difference)

Result

```
SELECT * FROM bridge
WHERE id NOT IN ( SELECT id FROM chess );
```

MINUS CLAUSE

- SELECT expression1, expression2, ... expression_n FROM tables [WHERE conditions]

  MINUS

  SELECT expression1, expression2, ... expression_n FROM tables [WHERE conditions];

SELECT supplier_id FROM suppliers

MINUS

SELECT supplier_id FROM orders;

8. SQL Joins

- A SQL JOIN clause combines records from two or more tables in a database.
- It creates a set that can be saved as a table or used as is.
- A JOIN is a means for combining fields from two tables by using values common to each.

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**.

  Example:

Emp

ID	State
1000	CA
1001	MA
1002	TN

Dept

ID	Division
1001	IT
1002	Sales
1003	Biotech

Types of Joins

- Equi-join
  - Inner join
  - Outer joins
    - Left outer join
    - Right outer joins
    - Full outer join
- Non Equi-join
- Cross join
- Self-join

EQUI JOIN Vs NON EQUI JOIN

- The SQL EQUI JOIN is a simple sql join uses the equal sign(=) as the comparison operator for the condition. It has two types - SQL Outer join and SQL Inner join

- The **SQL NON EQUI JOIN** is a join uses comparison operator other than the equal sign like >, <, >=, <= with the condition

INNER JOIN Vs OUTER JOIN

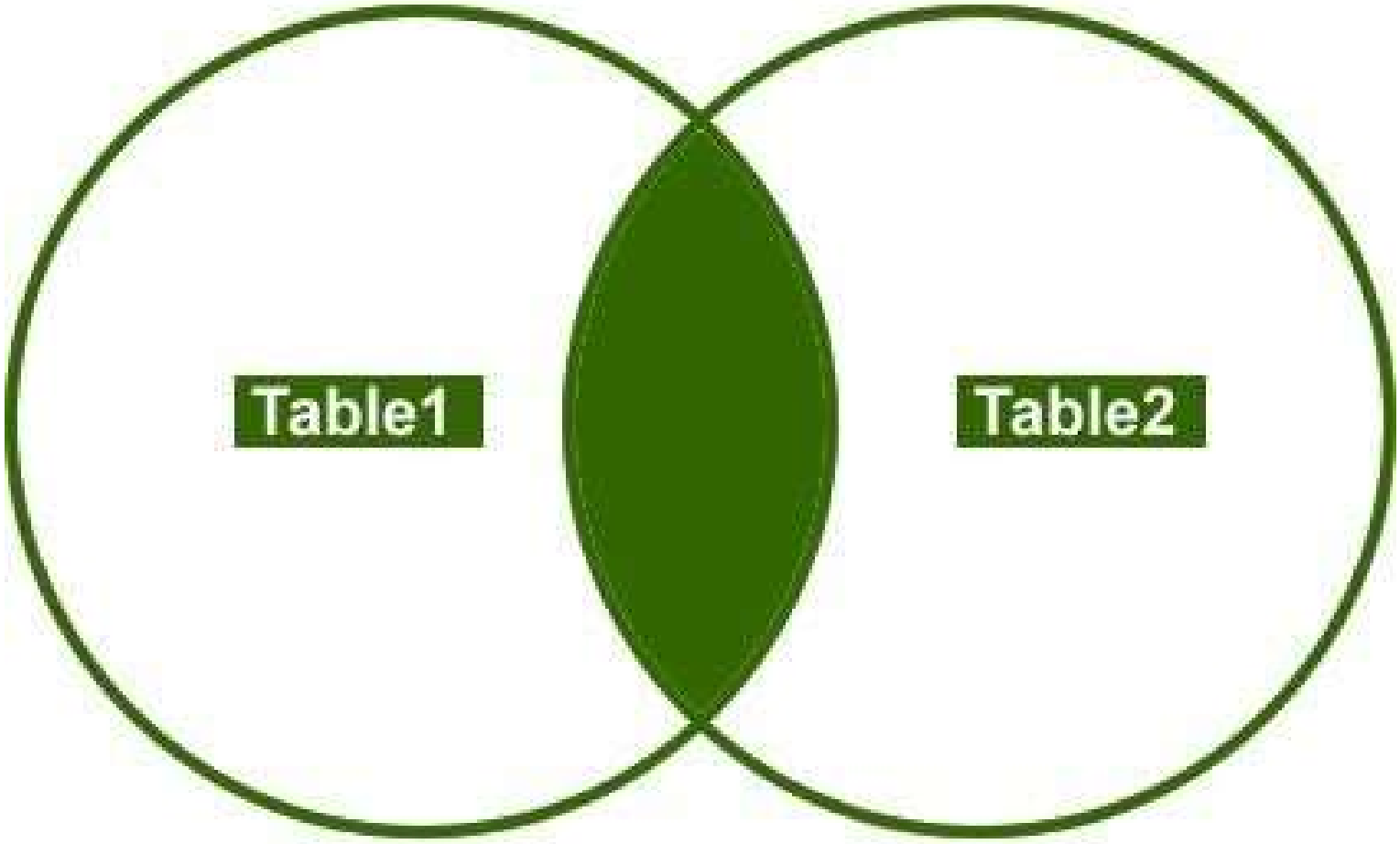
SQL INNER JOIN

- This type of EQUI JOIN returns all rows from tables where the key record of one table is equal to the key records of another table.

SQL OUTER JOIN

- This type of EQUI JOIN returns all rows from one table and only those rows from the secondary table where the joined condition is satisfying i.e. the columns are equal in both tables.

INNER JOIN




```
SELECT *
FROM Table1 t1
INNER JOIN Table2 t2
        ON t1.Col1 = t2.Col1
```


(C) http://blog.SQLAuthority.com

INNER JOIN

```
SELECT *
FROM table1, table2
WHERE table1.comcol = table2.comcol
```

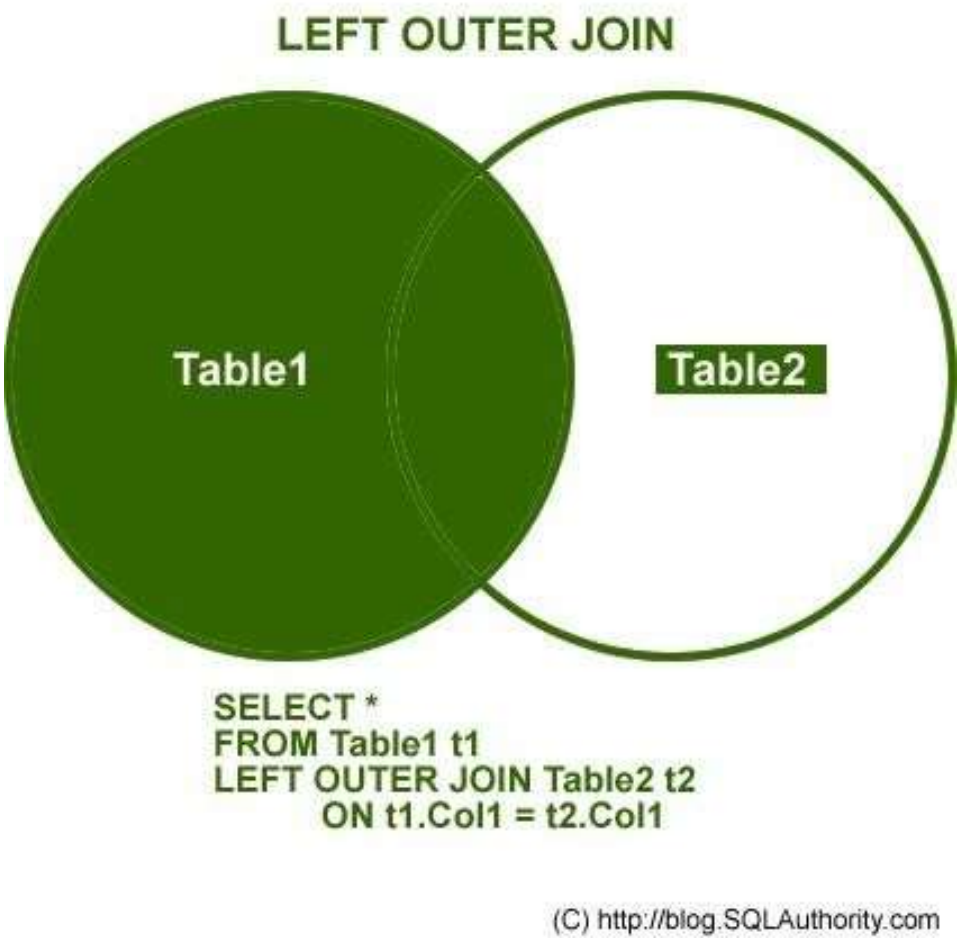
```
SELECT a.comcol, a.col1, b.col2, expr1, expr2
FROM table1 a, table2 b
WHERE a.comcol = b.comcol
```

```
SELECT a.comcol, a.col1, b.col2, expr1, expr2
FROM table1 a INNER JOIN table2 b
ON a.comcol = b.comcol
```

Outer Join

- Left outer join
- Right outer joins
- Full outer join

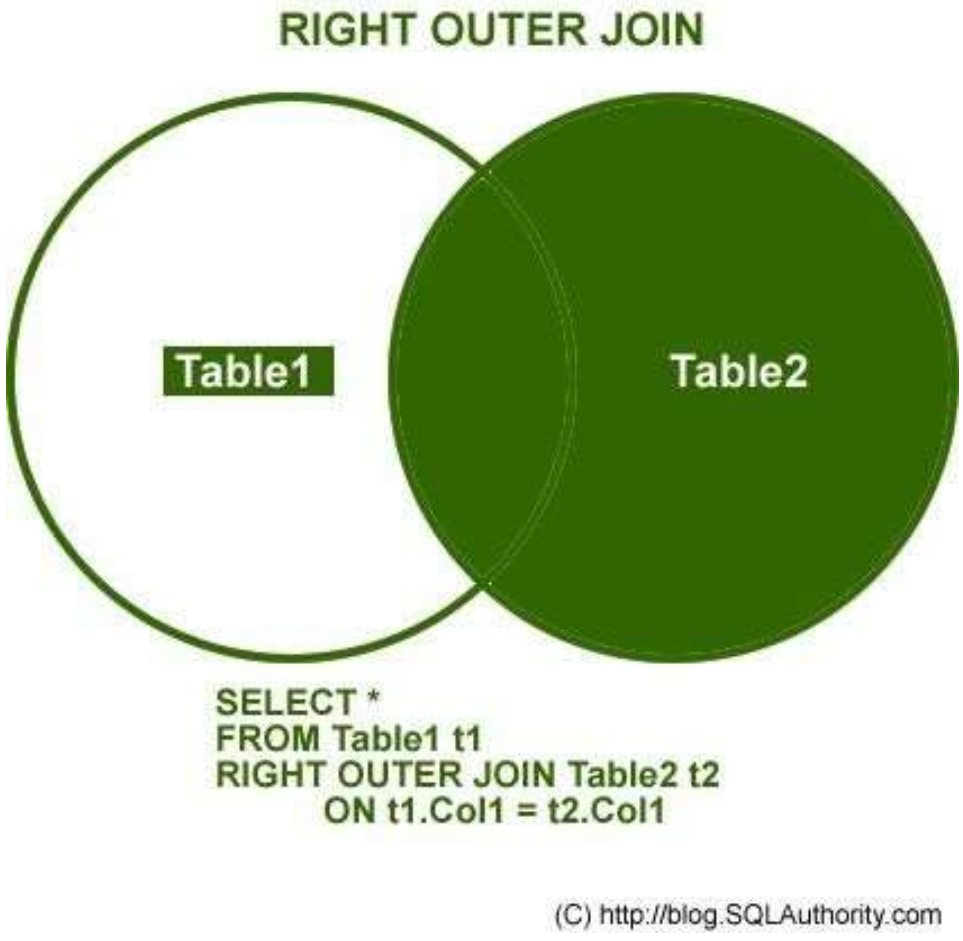
SQL JOINS



- **left outer join** = left join

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

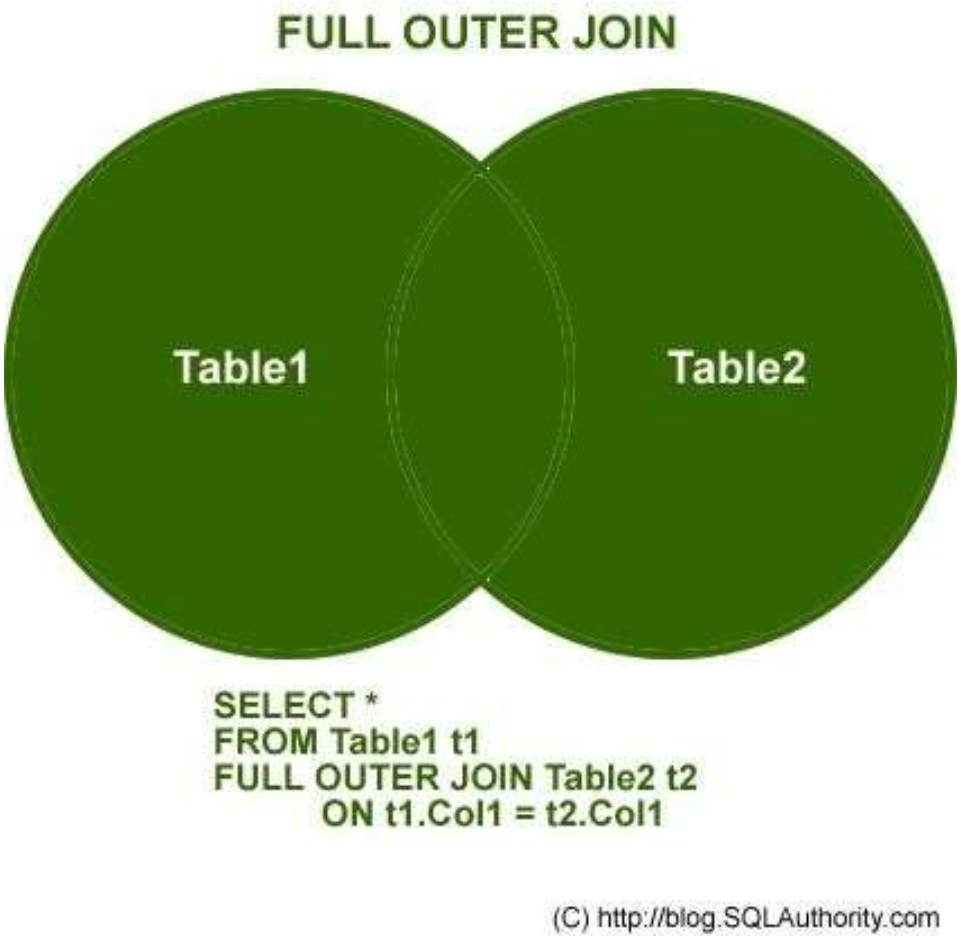
SQL JOINS



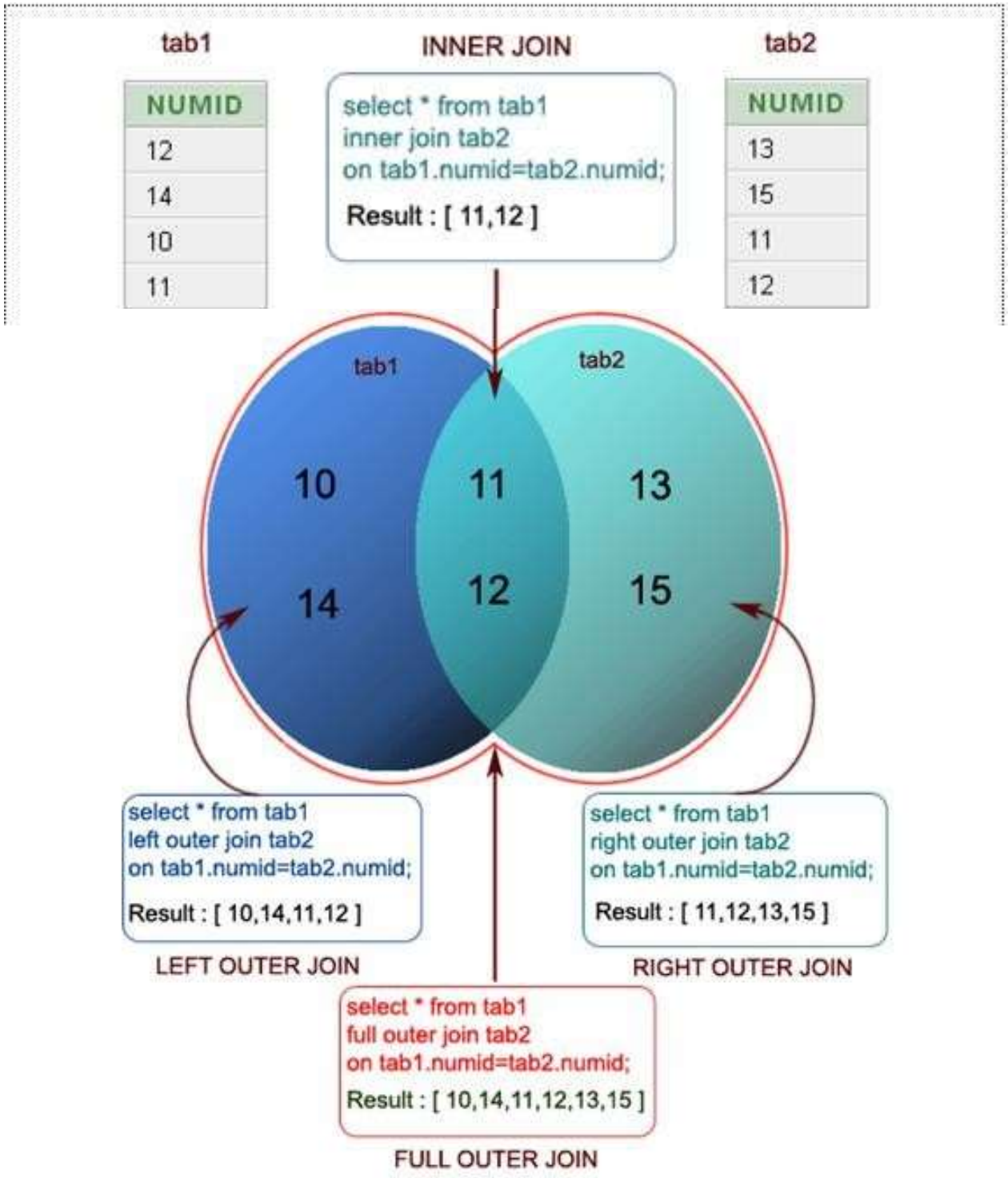
- **Right outer join** = right join

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

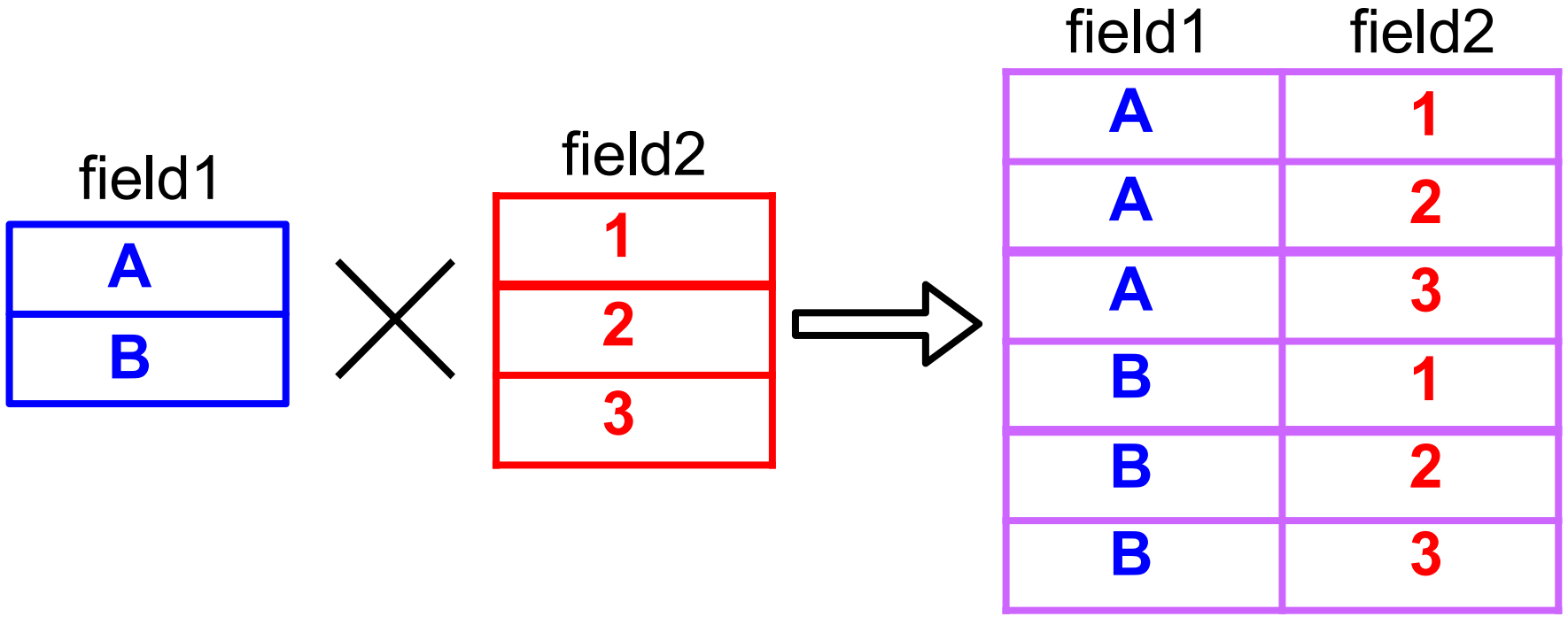
SQL JOINS




```
SELECT  *
FROMemp Full join dept
on emp.id = dept.id;
```


tab1
INNER JOIN
tab2
NUMID
12
14
10
11
select * from tab1
inner join tab2
on tab1.numid=tab2.numid;
Result : [11,12]
NUMID
13
15
11
12
tab1
tab2
10
11
13
14
12
15
select * from tab1
left outer join tab2
on tab1.numid=tab2.numid;
Result : [10,14,11,12]
LEFT OUTER JOIN
select * from tab1
right outer join tab2
on tab1.numid=tab2.numid;
Result : [11,12,13,15]
RIGHT OUTER JOIN
select * from tab1
full outer join tab2
on tab1.numid=tab2.numid;
Result : [10,14,11,12,13,15]
FULL OUTER JOIN

Cross Join



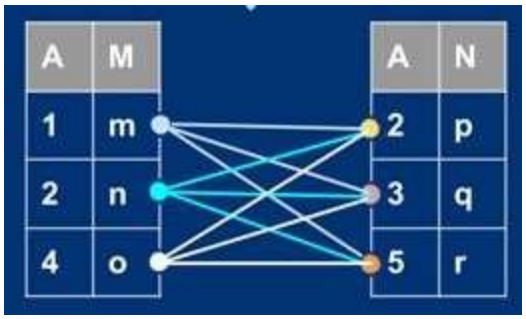
CROSS JOIN

Student

ID	Name
123	John
124	Mary
125	Mark
126	Jane

Enrolment

ID	Code
123	DBS
124	PRG
124	DBS
126	PRG




```
SELECT * FROM
   Student CROSS JOIN
   Enrolment
```

ID	Name	ID	Code
123	John	123	DBS
124	Mary	123	DBS
125	Mark	123	DBS
126	Jane	123	DBS
127	John	124	PRG
128	Mary	124	PRG
129	Mark	124	PRG
130	Jane	124	PRG
131	John	124	DBS
132	Mary	124	DBS

SELF JOIN

- **Joining the table itself** called self join.
- Self join is used to retrieve the records having some relation or similarity with other records in the same table.
- Here we need to use aliases for the same table to set a self join between single table and retrieve records satisfying the condition in where clause.

SELF JOIN - EXAMPLE

```
select e2.EmpName,e1.EmpName as 'Manager'
from EmployeeDetails e1
INNER JOIN EmployeeDetails e2
on e1.EmpID=e2.EmpMgrID
```

9. DCL: Data Control Language

- Controlling Access to database objects such as tables and views
- DCL commands
  - **GRANT** – to allow specified users to perform specified tasks
  - **REVOKE** - to cancel previously granted or denied permissions

GRANT

- Syntax
  - GRANT <privileges> ON <object name>
    TO <grantee1> ,<grantee2> ... |PUBLIC |role-name [
    WITH GRANT OPTION]
  - WITH GRANT OPTION: allows the grantee to further grant privileges
  - PUBLIC is used to grant access rights to all users
  - ROLES - set of privileges grouped together.
- Example :

Granting “Mary” the access to Table “student” (for inserting, updating and deleting)

- GRANT INSERT, UPDATE, DELETE ON Student TO Mary;

REVOKE

- Syntax
- REVOKE <privileges>
  ON <object_name>
  FROM user_name |PUBLIC |role-name

- REVOKE DELETE ON Student FROM Mary;

THANK YOU