TOPIC 4:

STRUCTURED QUERY LANGUAGE (SQL)

CLO 1 : Apply fundamental of Database Management System (DBMS), relational data model and normalization concepts in database development process.

CLO 2 : Show a well-structured database using the database query to manipulate a database with an appropriate commercial Database Management System.

At the end of this session, students should be able to:

4.1 Apply SQL commands to a database

Definition of SQL

- SQL, short for Structured Query Language is a simple programming language used for accessing and managing data in relational databases.

- SQL uses simple commands like select, insert, update and delete to manipulate rows of data.

Uses of SQL

- Allow user to create database and relation structure
- Perform basic data management tasks, such as the insertion, modification, and deletion of data from the relations

Importance of SQL

- SQL is the first and so far only standard database language to gain wide acceptance
- Newly every major vendor provide database products based on SQL or with SQL interface
- It has become part of application architectures

Data Definition Language (DDL)

- It is mainly used to create files, databases, data dictionary and tables within databases.
- It also used to specify the structure of each table, set of associate values with each attribute, integrity constraints, security and authorization information for each table and physical storage structure of each table.

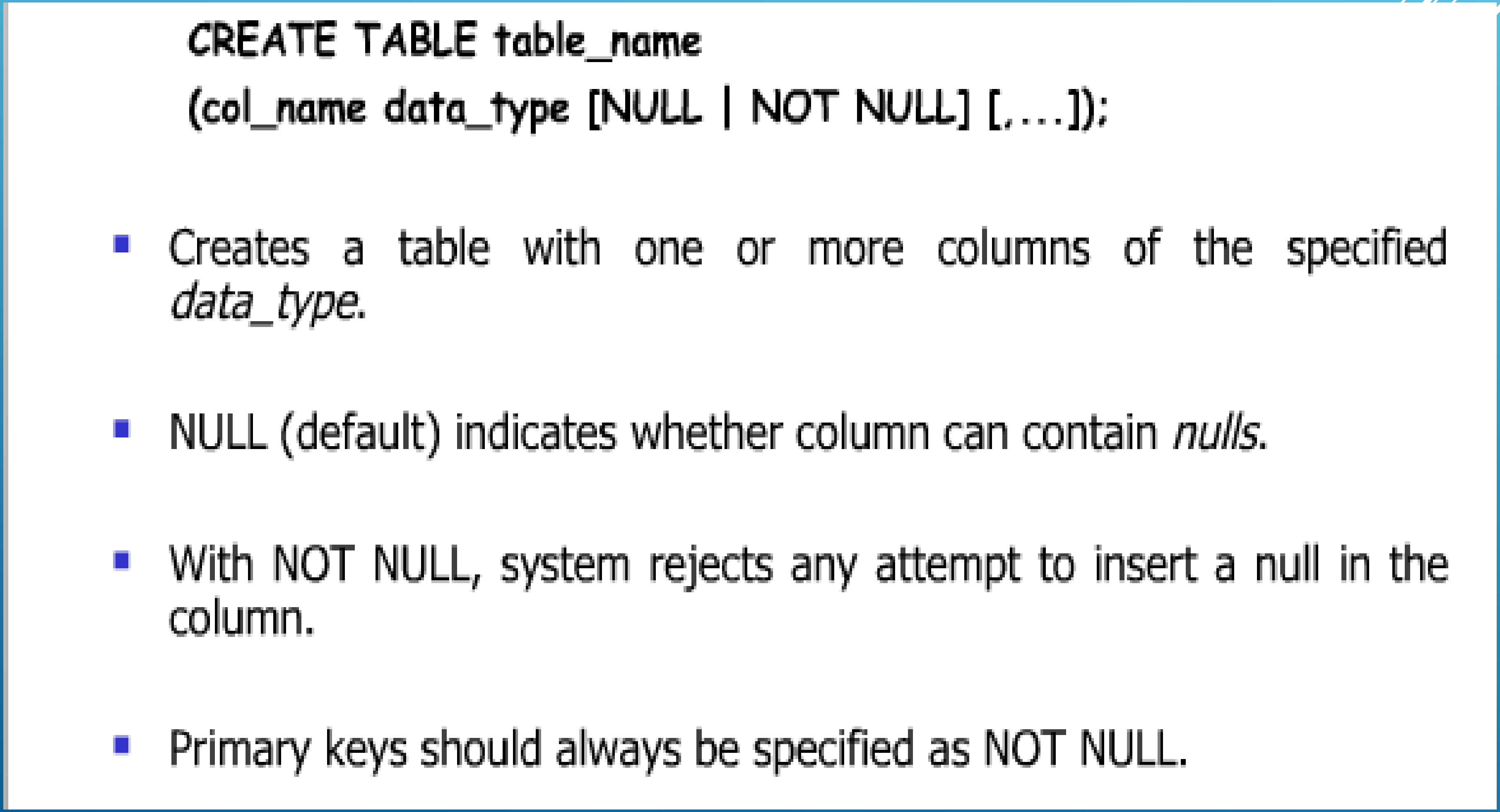
Data Manipulation Language (DML)

- It is a language that provides a set of operations to support the basic data manipulation operation on the data held in the databases.
- It allows users to insert, update, delete and retrieve data from the database.
- The part of DML that involves data retrieval is called a query language.

4 Basic DDL Commands

1. Create schema objects CREATE
2. Alter schema objects ALTER
3. Delete Schema Objects DROP
4. Rename schema objects RENAME

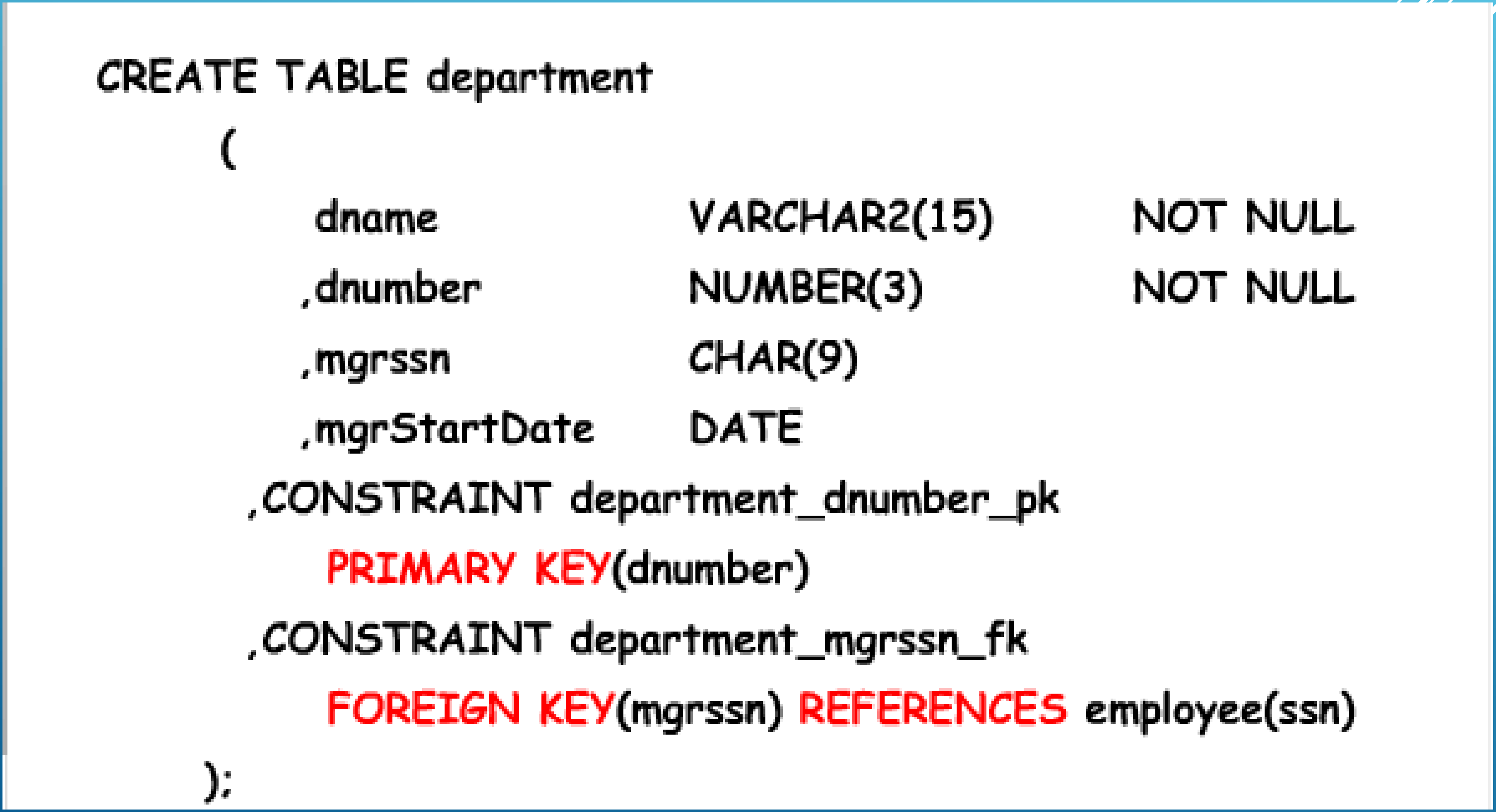
CREATE Commands



```
CREATE TABLE table_name
(col_name data_type [NULL | NOT NULL] [,...]);
```

- Creates a table with one or more columns of the specified *data_type*.
- NULL (default) indicates whether column can contain *nulls*.
- With NOT NULL, system rejects any attempt to insert a null in the column.
- Primary keys should always be specified as NOT NULL.

CREATE Commands



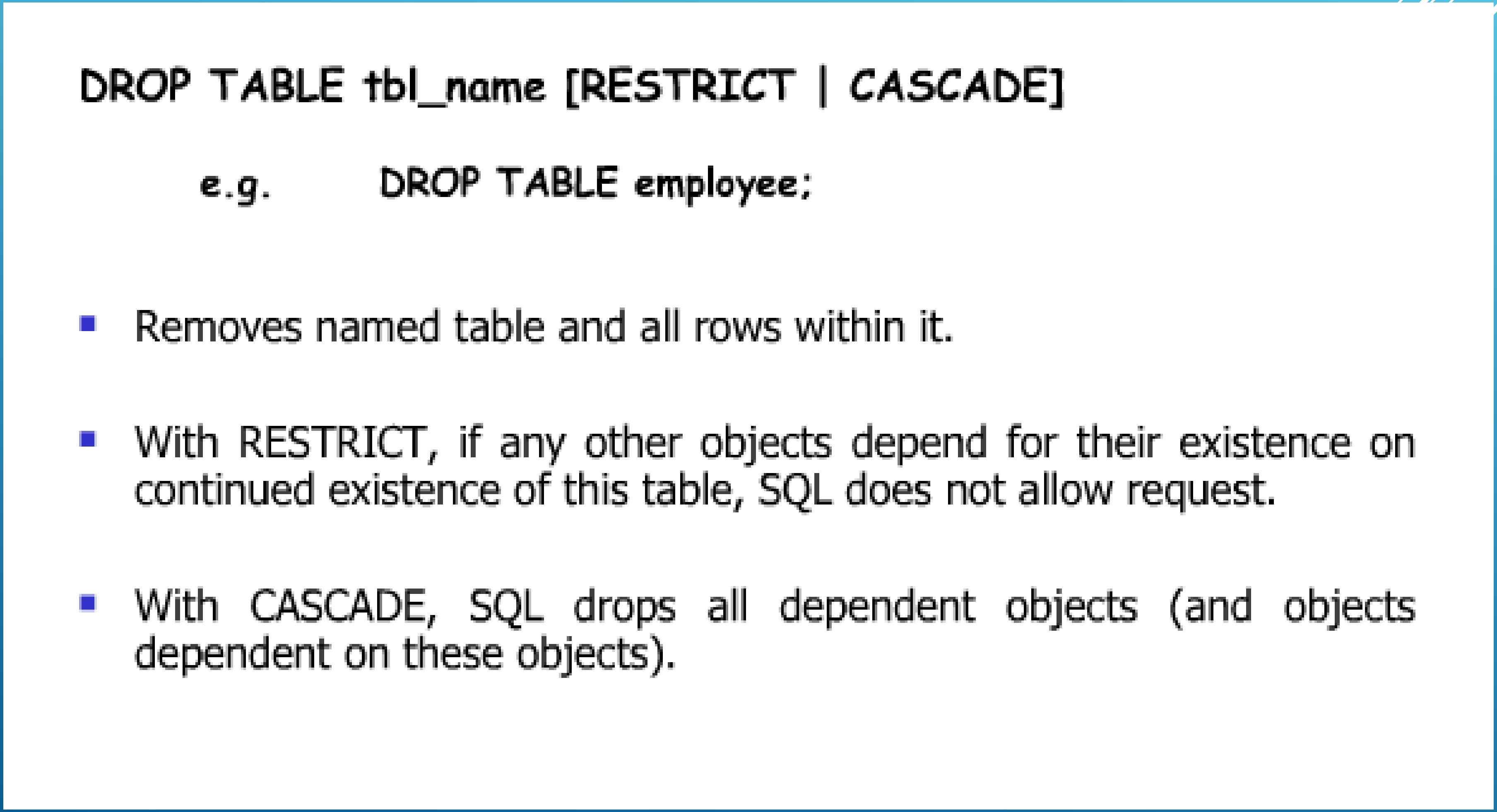
```
CREATE TABLE department
      (
          dname               VARCHAR2(15)        NOT NULL
         ,dnumber             NUMBER(3)           NOT NULL
         ,mgrssn              CHAR(9)
         ,mgrStartDate        DATE
       ,CONSTRAINT department_dnumber_pk
           PRIMARY KEY(dnumber)
       ,CONSTRAINT department_mgrssn_fk
           FOREIGN KEY(mgrssn) REFERENCES employee(ssn)
      );
```

ALTER Commands

- The ALTER command is a schema modification command.
- It is used to add or drop a column, change a column definition, add or drop table constraints.
- Example:

```
ALTER TABLE COMPANY.EMPLOYEE
MODIFY(lname VARCHAR2(30));
```

DROP Commands



```
DROP TABLE tbl_name [RESTRICT | CASCADE]

     e.g.     DROP TABLE employee;
```

- Removes named table and all rows within it.
- With RESTRICT, if any other objects depend for their existence on continued existence of this table, SQL does not allow request.
- With CASCADE, SQL drops all dependent objects (and objects dependent on these objects).

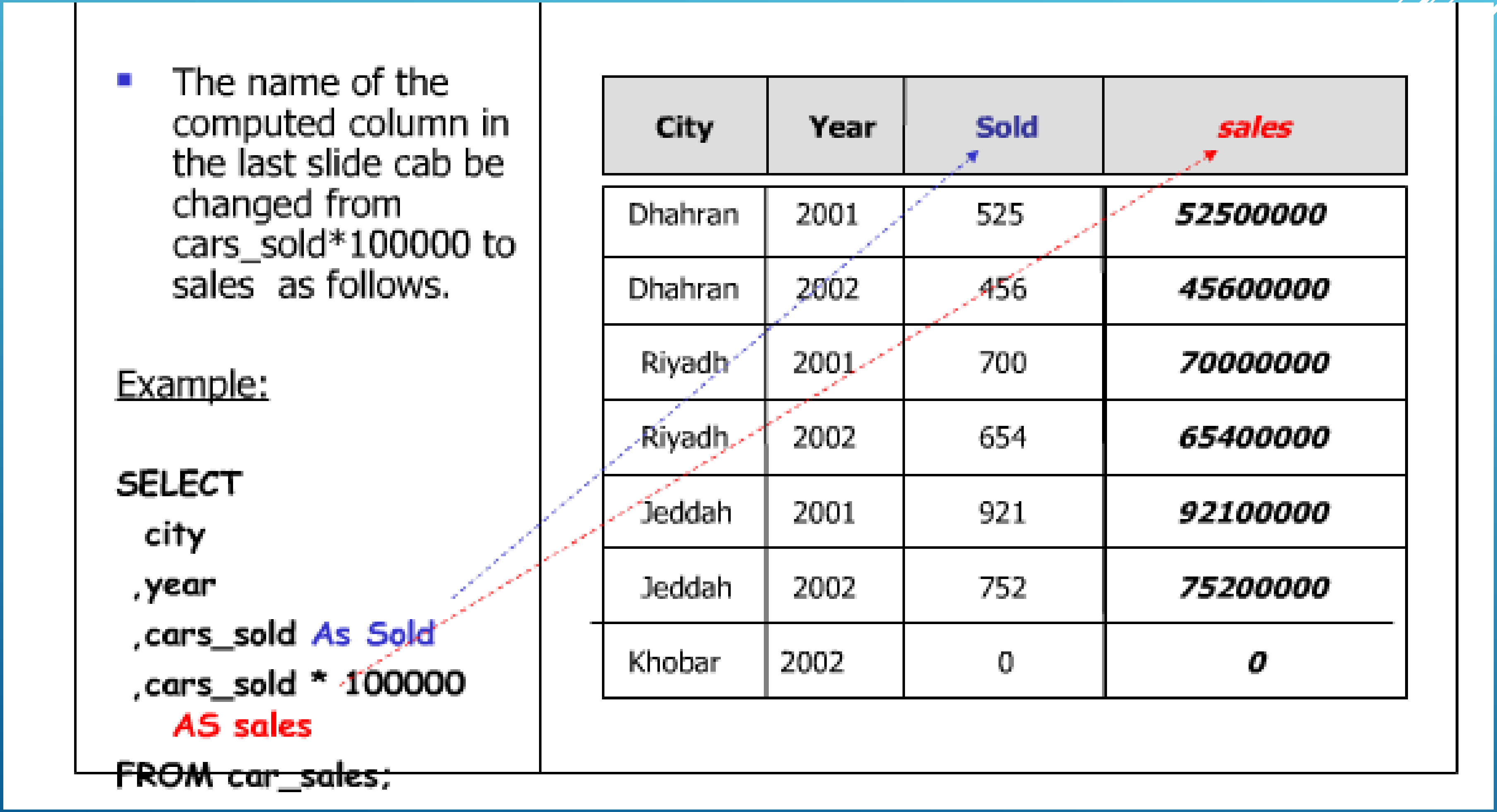
RENAME Commands

- Used to change the name of the table or a database object.

```
RENAME old_table_name TO new_table_name

eg:     RENAME employee TO my_employee;
```

RENAME Commands



- The name of the computed column in the last slide cab be changed from cars_sold*100000 to sales as follows.

Example:

```
SELECT
  city
 ,year
 ,cars_sold As Sold
 ,cars_sold * 100000
     AS sales
FROM car_sales;
```

City	Year	Sold	sales
Dhahran	2001	525	***52500000***
Dhahran	2002	456	***45600000***
Riyadh	2001	700	***70000000***
Riyadh	2002	654	***65400000***
Jeddah	2001	921	***92100000***
Jeddah	2002	752	***75200000***
Khobar	2002	0	***0***

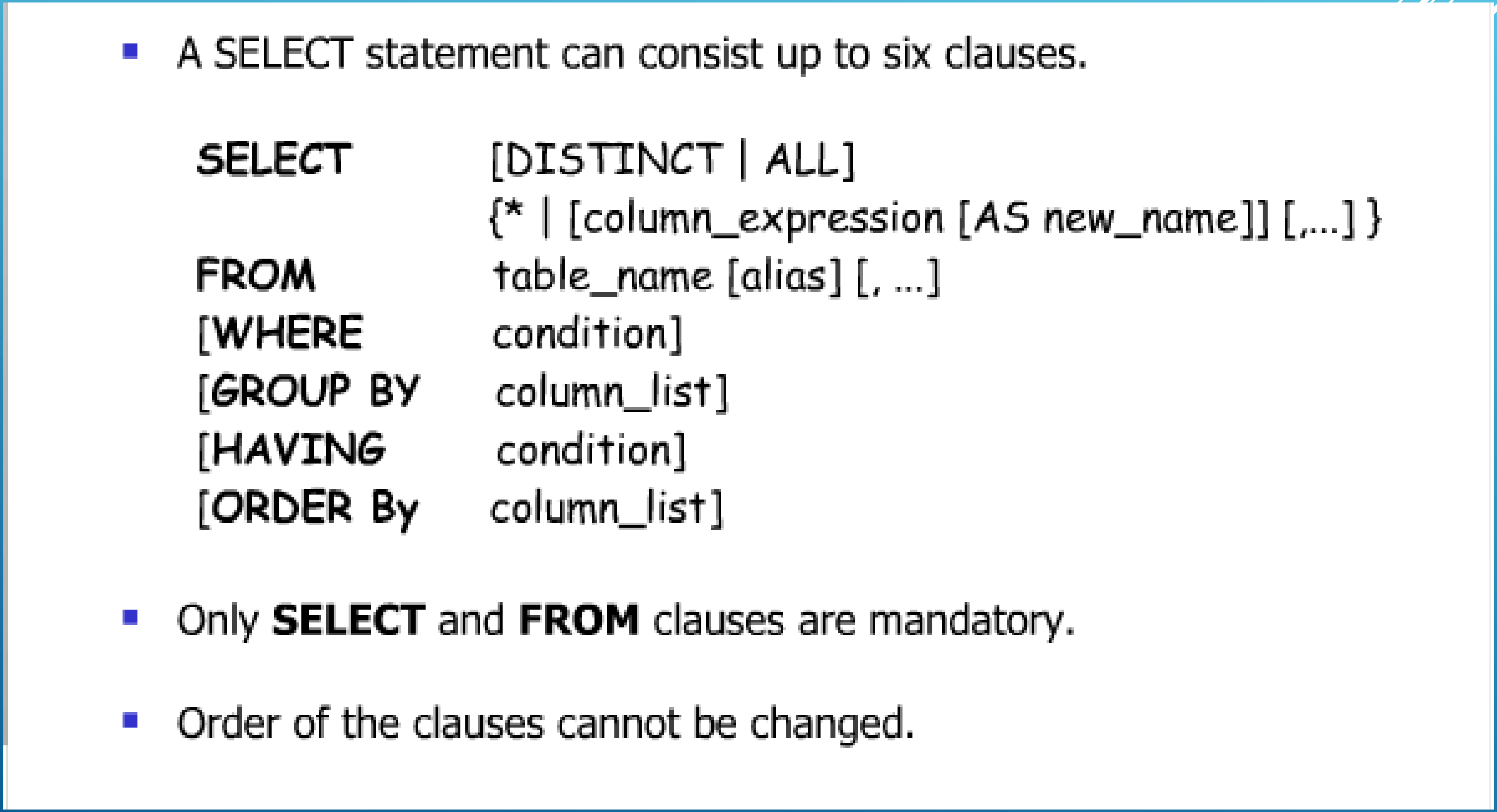
4 Basic DML Commands

1. Retrieve data SELECT
2. Remove rows DELETE
3. Add new row INSERT
4. Change column value UPDATE

SELECT Commands

- SQL has only one statement for retrieving information from a database called the SELECT statement.
- SQL SELECT statement is different from that of Relational Algebra.
- An important distinction between SQL and formal relational model is that SQL allows duplicate rows. Hence an SQL table is not a set but a multiset (some times called a bag) of tuples.

SELECT Commands



- A SELECT statement can consist up to six clauses.

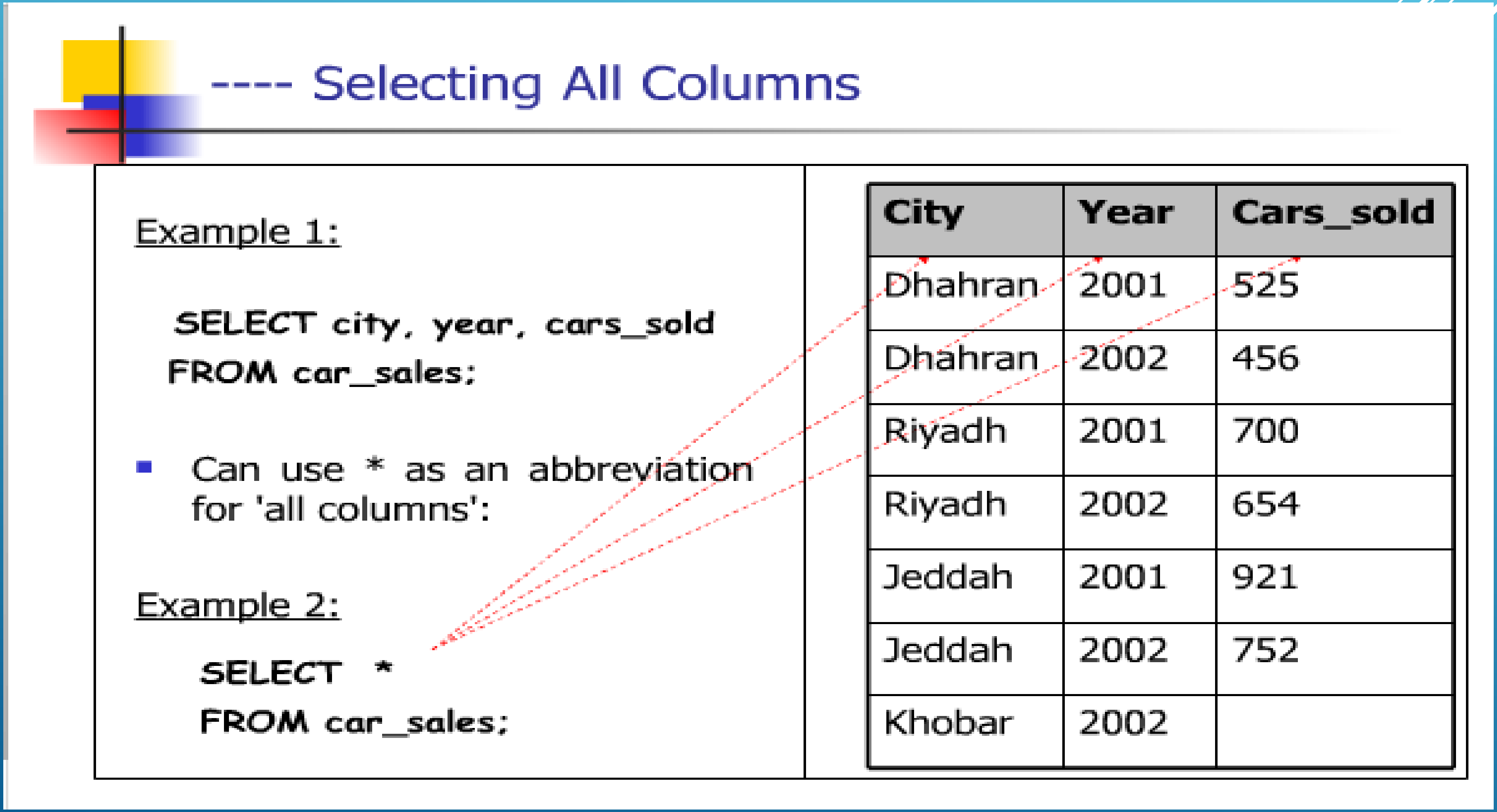
```
SELECT        [DISTINCT | ALL]
              {* | [column_expression [AS new_name]] [,...] }
FROM          table_name [alias] [, ...]
[WHERE        condition]
[GROUP BY     column_list]
[HAVING       condition]
[ORDER By     column_list]
```

- Only **SELECT** and **FROM** clauses are mandatory.
- Order of the clauses cannot be changed.

SELECT Commands

- **FROM** Specifies table(s) to be used.
- **WHERE** Filters rows.
- **GROUP BY** Forms groups of rows with same column value.
- **HAVING** Filters groups subject to some condition.
- **SELECT** Specifies which columns are to appear in output.
- **ORDER BY** Specifies the order of the output.

SELECT Commands



---- Selecting All Columns

Example 1:

```
SELECT city, year, cars_sold
FROM car_sales;
```

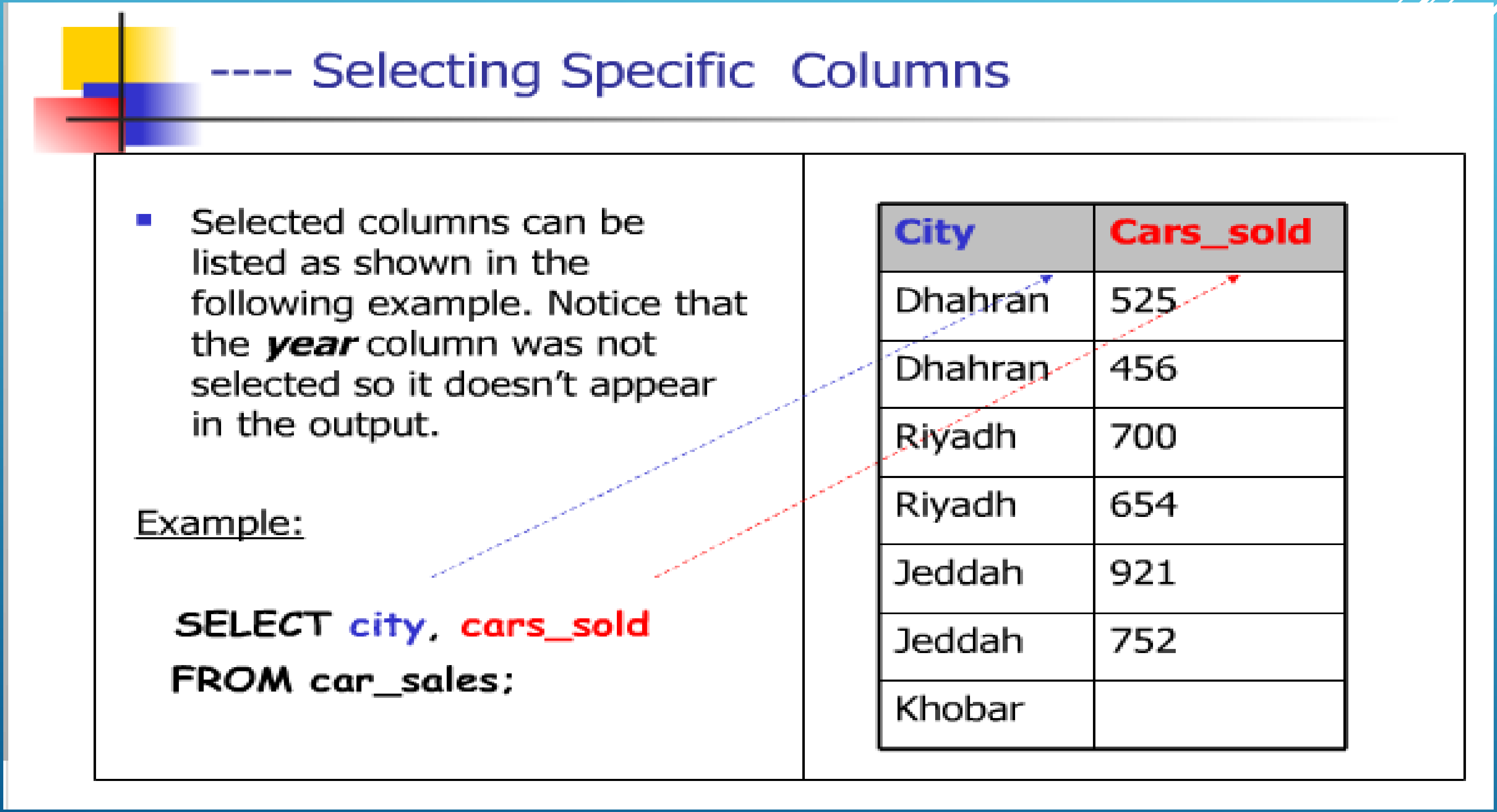
- Can use * as an abbreviation for 'all columns':

Example 2:

```
SELECT  *
FROM car_sales;
```

City	Year	Cars_sold
Dhahran	2001	525
Dhahran	2002	456
Riyadh	2001	700
Riyadh	2002	654
Jeddah	2001	921
Jeddah	2002	752
Khobar	2002	

SELECT Commands



---- Selecting Specific Columns

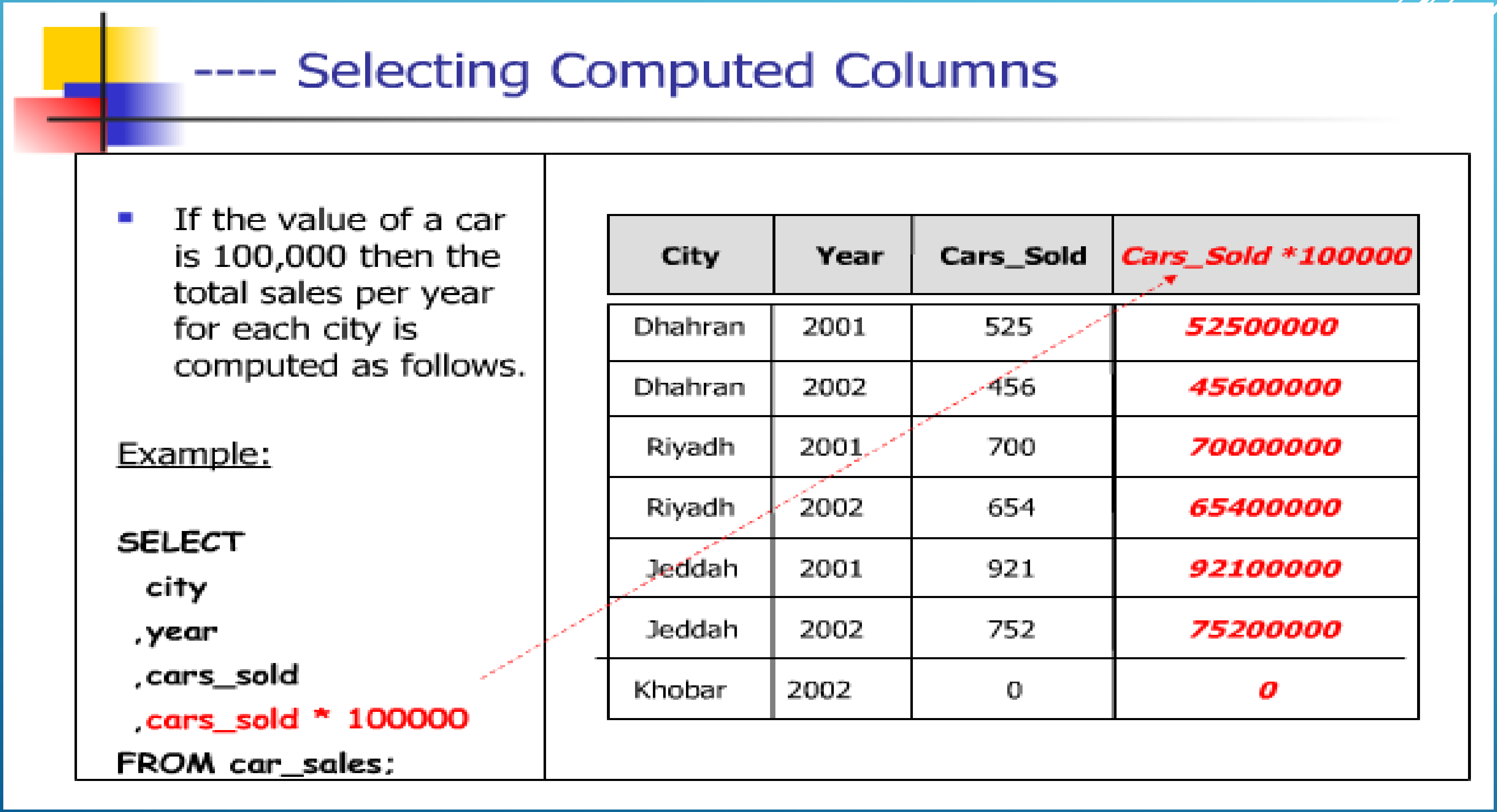
- Selected columns can be listed as shown in the following example. Notice that the ***year*** column was not selected so it doesn't appear in the output.

Example:

```
SELECT city, cars_sold
FROM car_sales;
```

City	Cars_sold
Dhahran	525
Dhahran	456
Riyadh	700
Riyadh	654
Jeddah	921
Jeddah	752
Khobar	

SELECT Commands



---- Selecting Computed Columns

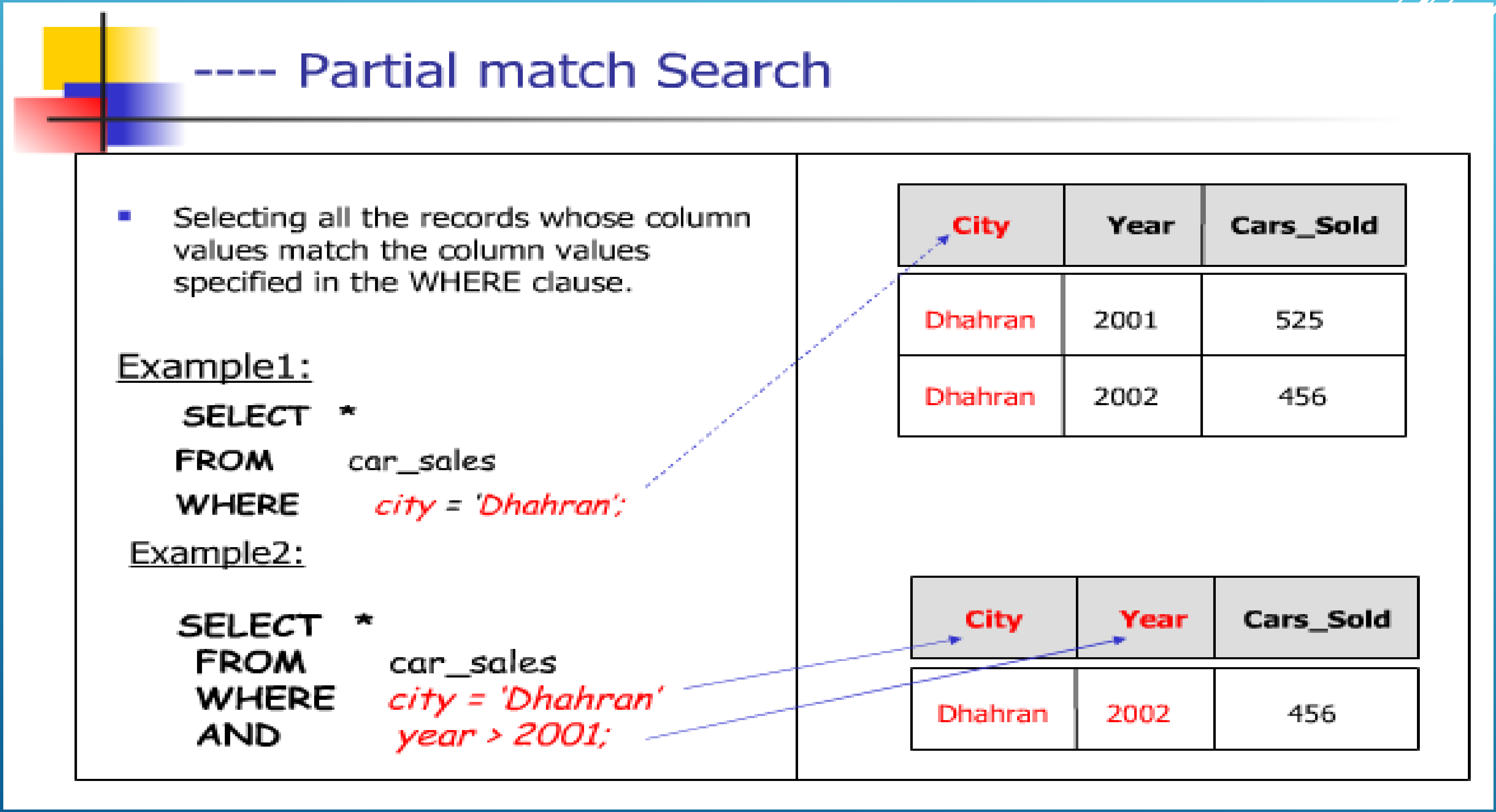
- If the value of a car is 100,000 then the total sales per year for each city is computed as follows.

Example:

```
SELECT
   city
  ,year
  ,cars_sold
  ,cars_sold * 100000
FROM car_sales;
```

City	Year	Cars_Sold	Cars_Sold *100000
Dhahran	2001	525	52500000
Dhahran	2002	456	45600000
Riyadh	2001	700	70000000
Riyadh	2002	654	65400000
Jeddah	2001	921	92100000
Jeddah	2002	752	75200000
Khobar	2002	0	0

SELECT Commands



---- Partial match Search

- Selecting all the records whose column values match the column values specified in the WHERE clause.

Example1:

```
SELECT   *
FROM     car_sales
WHERE    city = 'Dhahran';
```

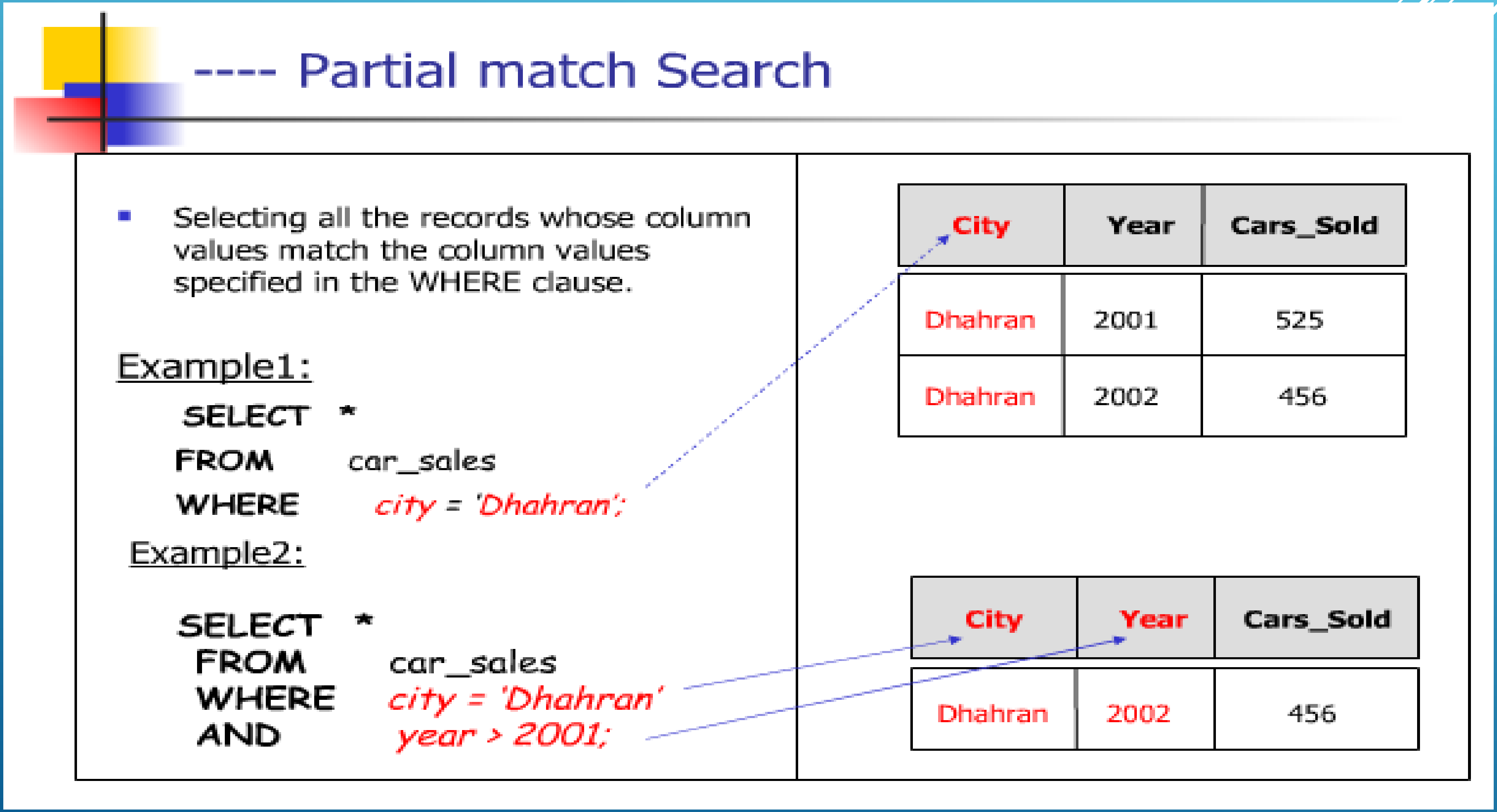
City	Year	Cars_Sold
Dhahran	2001	525
Dhahran	2002	456

Example2:

```
SELECT   *
 FROM     car_sales
 WHERE    city = 'Dhahran'
 AND       year > 2001;
```

City	Year	Cars_Sold
Dhahran	2002	456

SELECT Commands



---- Partial match Search

- Selecting all the records whose column values match the column values specified in the WHERE clause.

Example1:

```
SELECT  *
FROM    car_sales
WHERE   city = 'Dhahran';
```

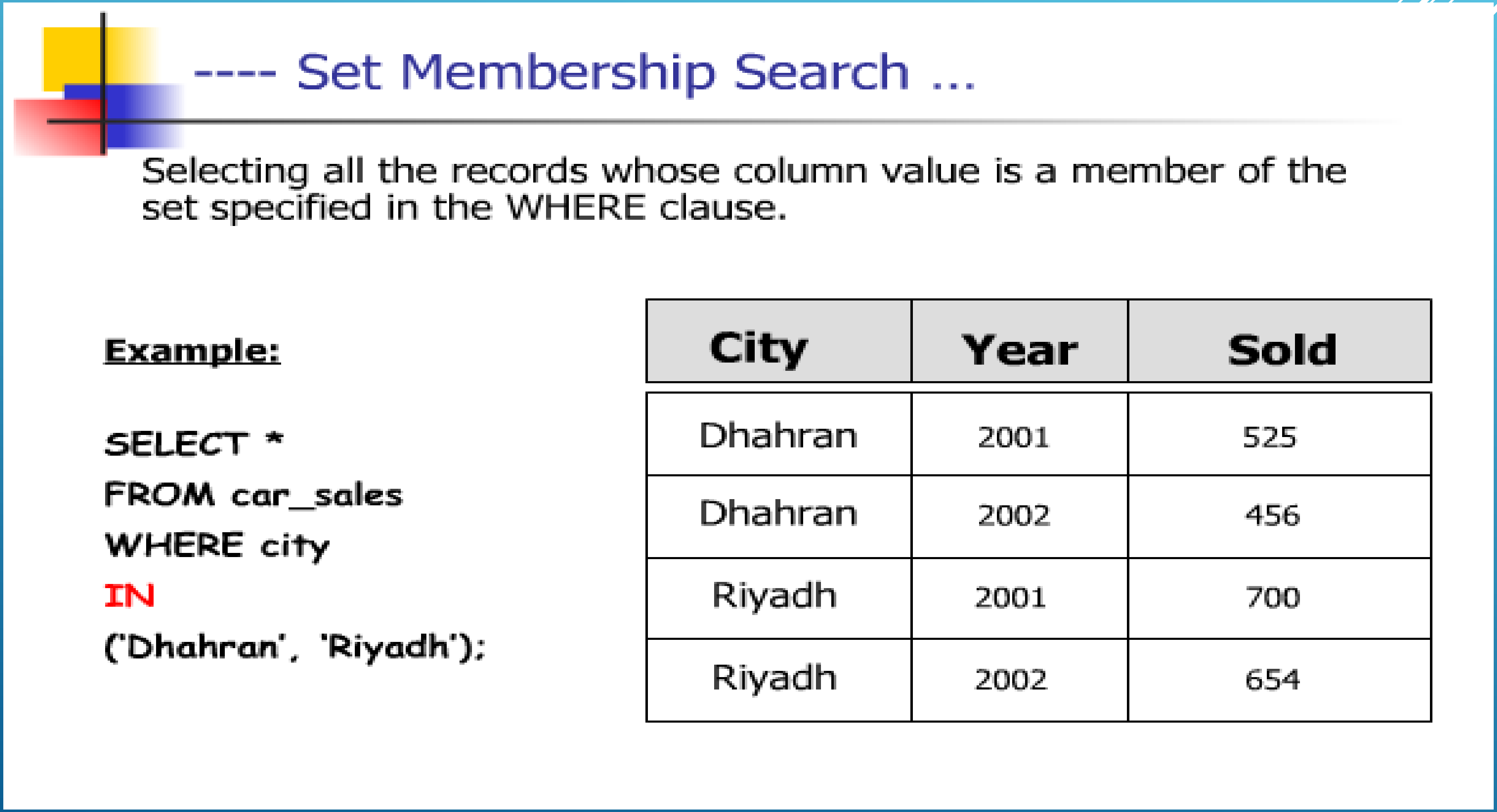
City	Year	Cars_Sold
Dhahran	2001	525
Dhahran	2002	456

Example2:

```
SELECT  *
FROM    car_sales
WHERE   city = 'Dhahran'
AND     year > 2001;
```

City	Year	Cars_Sold
Dhahran	2002	456

SELECT Commands



---- Set Membership Search ...

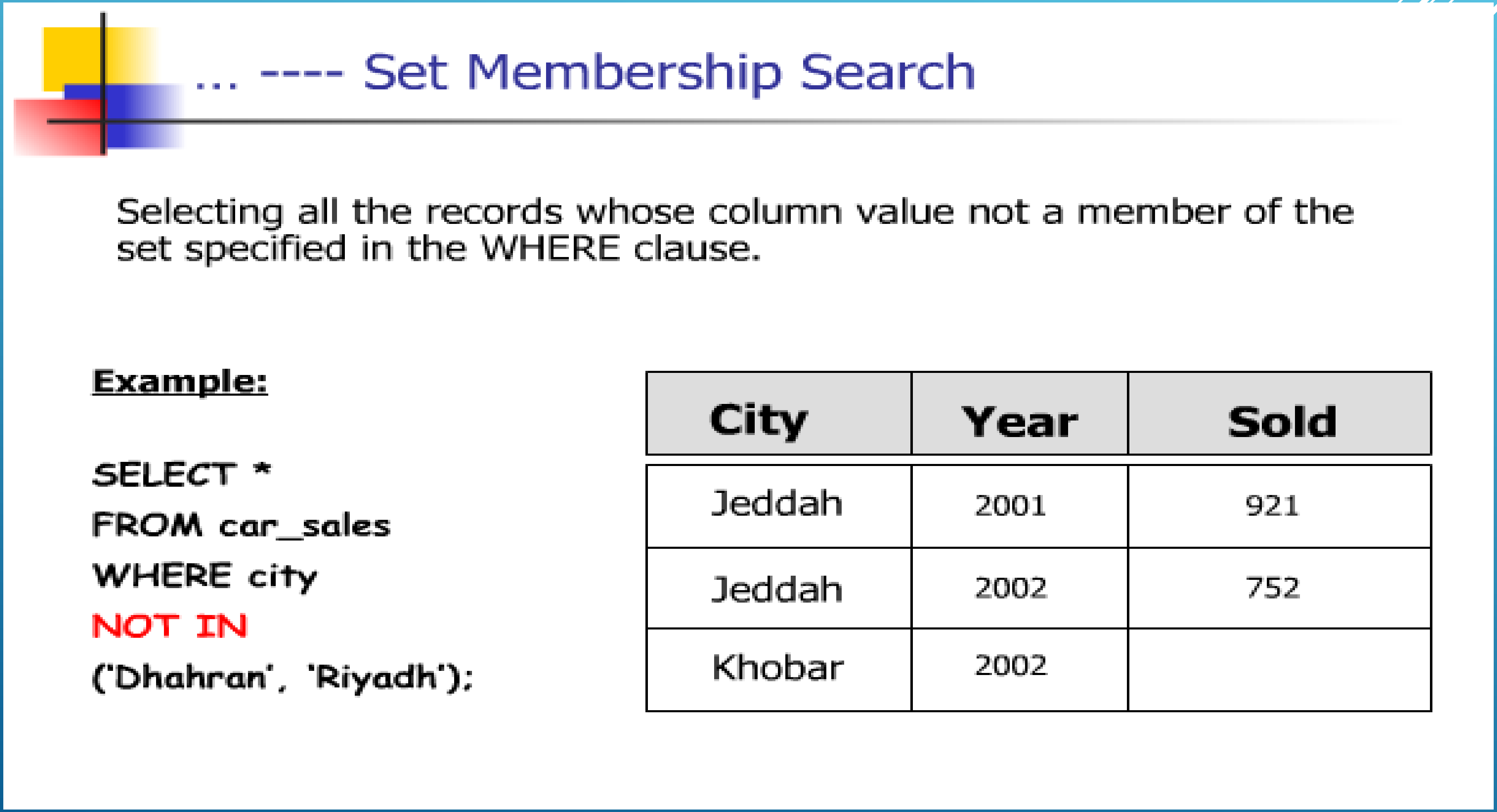
Selecting all the records whose column value is a member of the set specified in the WHERE clause.

Example:

```
SELECT *
FROM car_sales
WHERE city
IN
('Dhahran', 'Riyadh');
```

City	Year	Sold
Dhahran	2001	525
Dhahran	2002	456
Riyadh	2001	700
Riyadh	2002	654

SELECT Commands



… ---- Set Membership Search

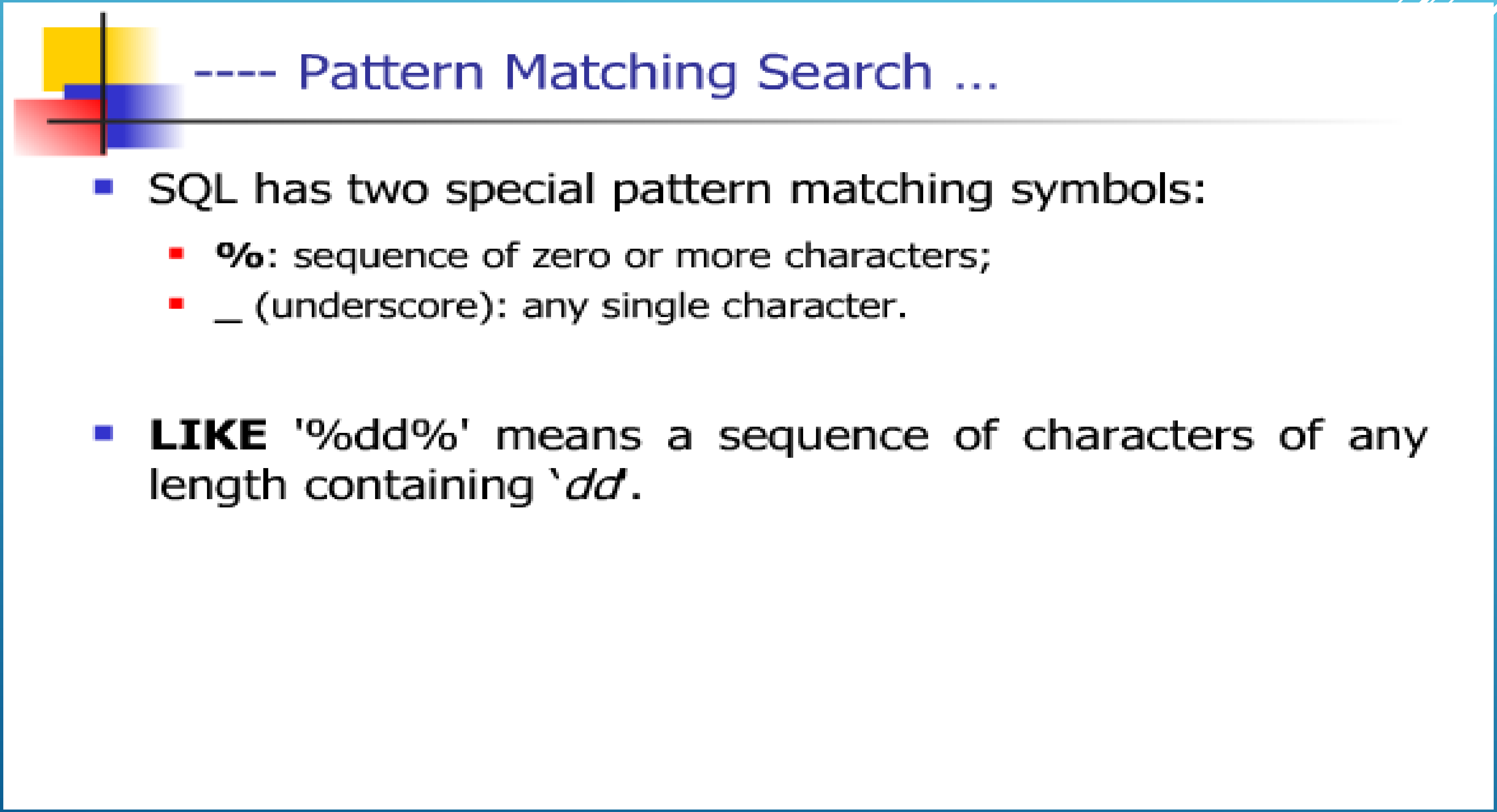
Selecting all the records whose column value not a member of the set specified in the WHERE clause.

Example:

```
SELECT *
FROM car_sales
WHERE city
NOT IN
('Dhahran', 'Riyadh');
```

City	Year	Sold
Jeddah	2001	921
Jeddah	2002	752
Khobar	2002	

SELECT Commands

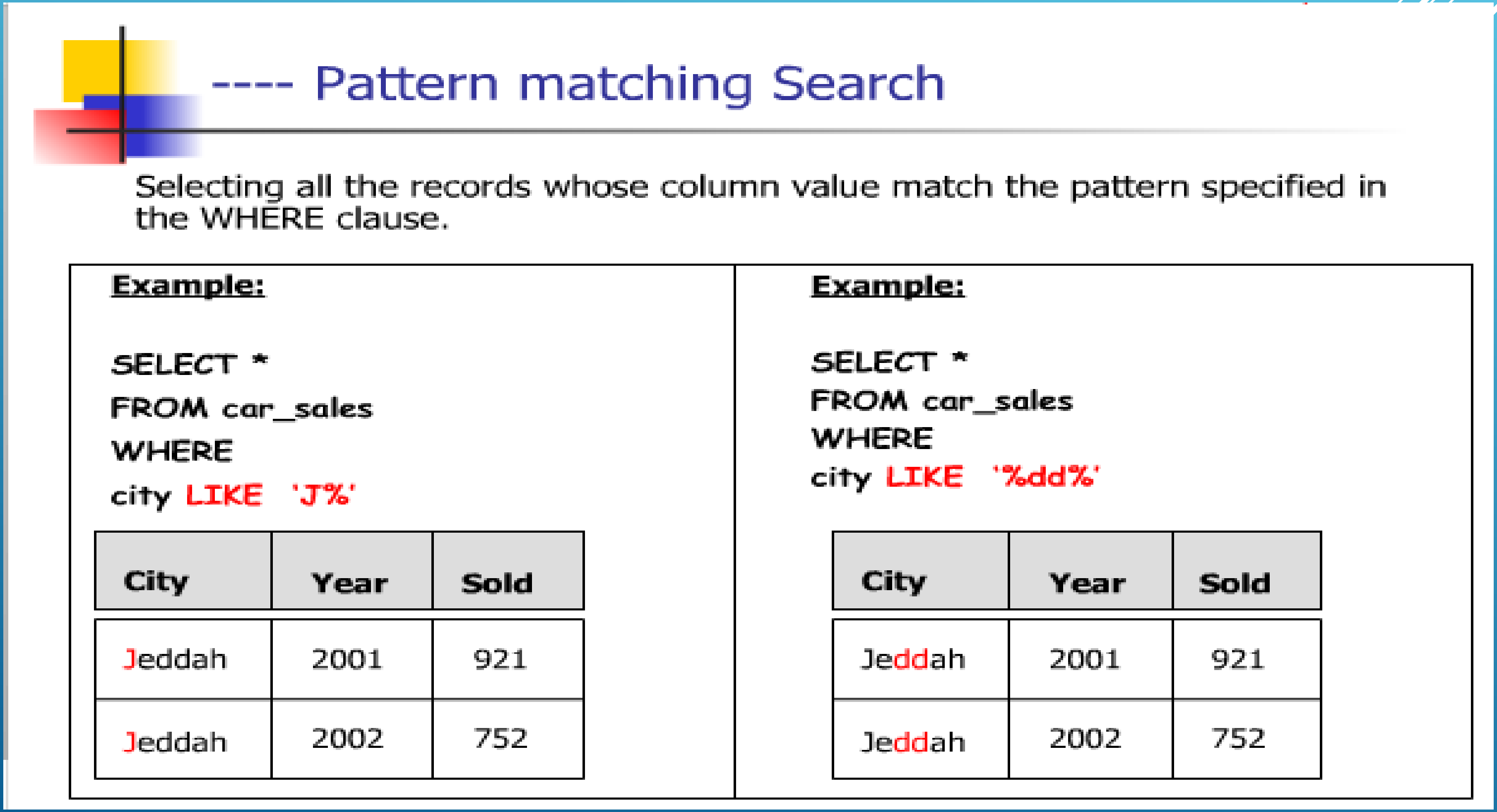


---- Pattern Matching Search ...

- SQL has two special pattern matching symbols:
  - **%**: sequence of zero or more characters;
  - _ (underscore): any single character.

- **LIKE** '%dd%' means a sequence of characters of any length containing '*dd*'.

SELECT Commands



---- Pattern matching Search

Selecting all the records whose column value match the pattern specified in the WHERE clause.

Example:

```
SELECT *
FROM car_sales
WHERE
city LIKE  'J%'
```

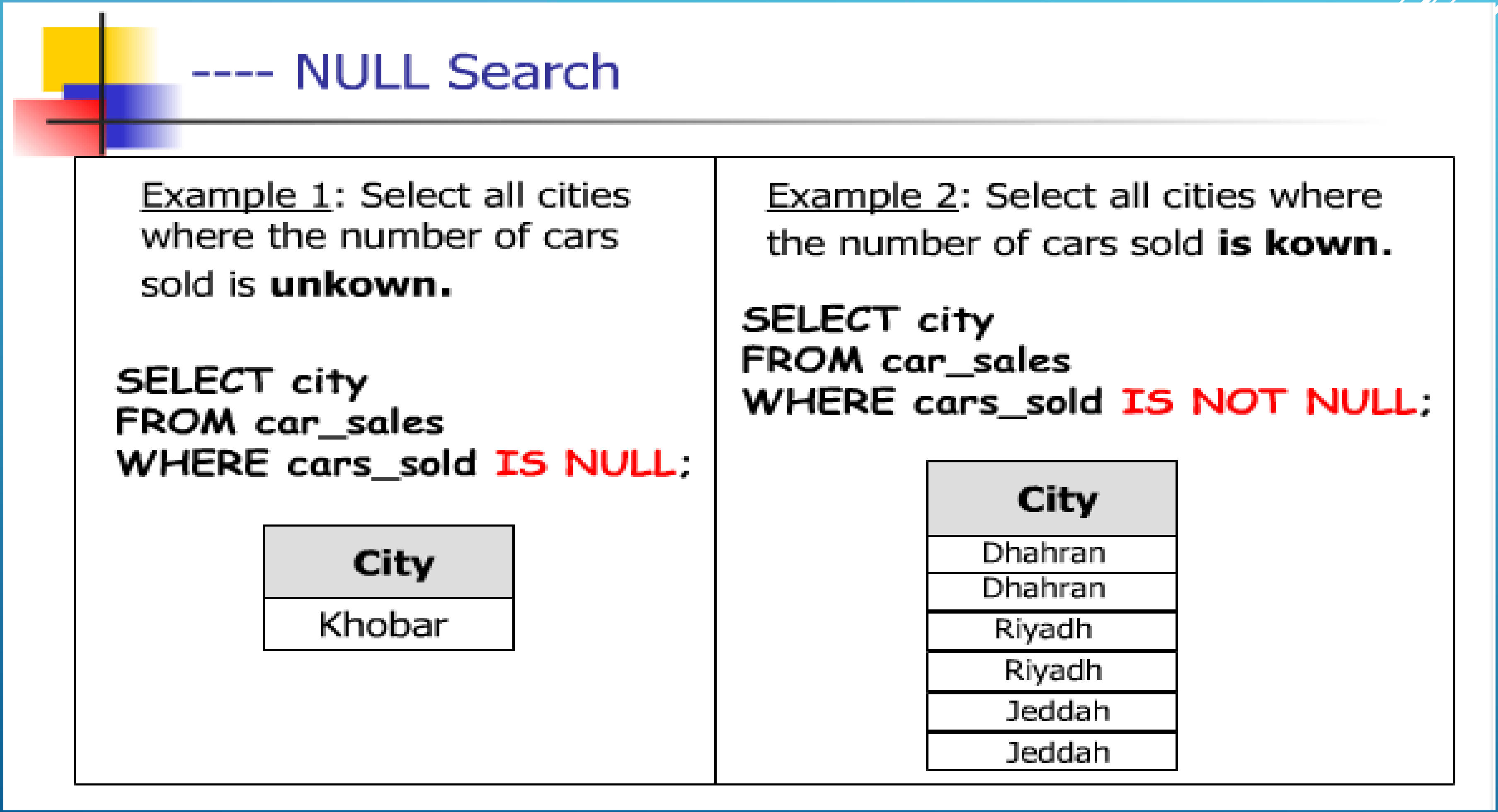
City	Year	Sold
Jeddah	2001	921
Jeddah	2002	752

Example:

```
SELECT *
FROM car_sales
WHERE
city LIKE  '%dd%'
```

City	Year	Sold
Jeddah	2001	921
Jeddah	2002	752

SELECT Commands



---- NULL Search

Example 1: Select all cities where the number of cars sold is **unkown.**

```
SELECT city
FROM car_sales
WHERE cars_sold IS NULL;
```

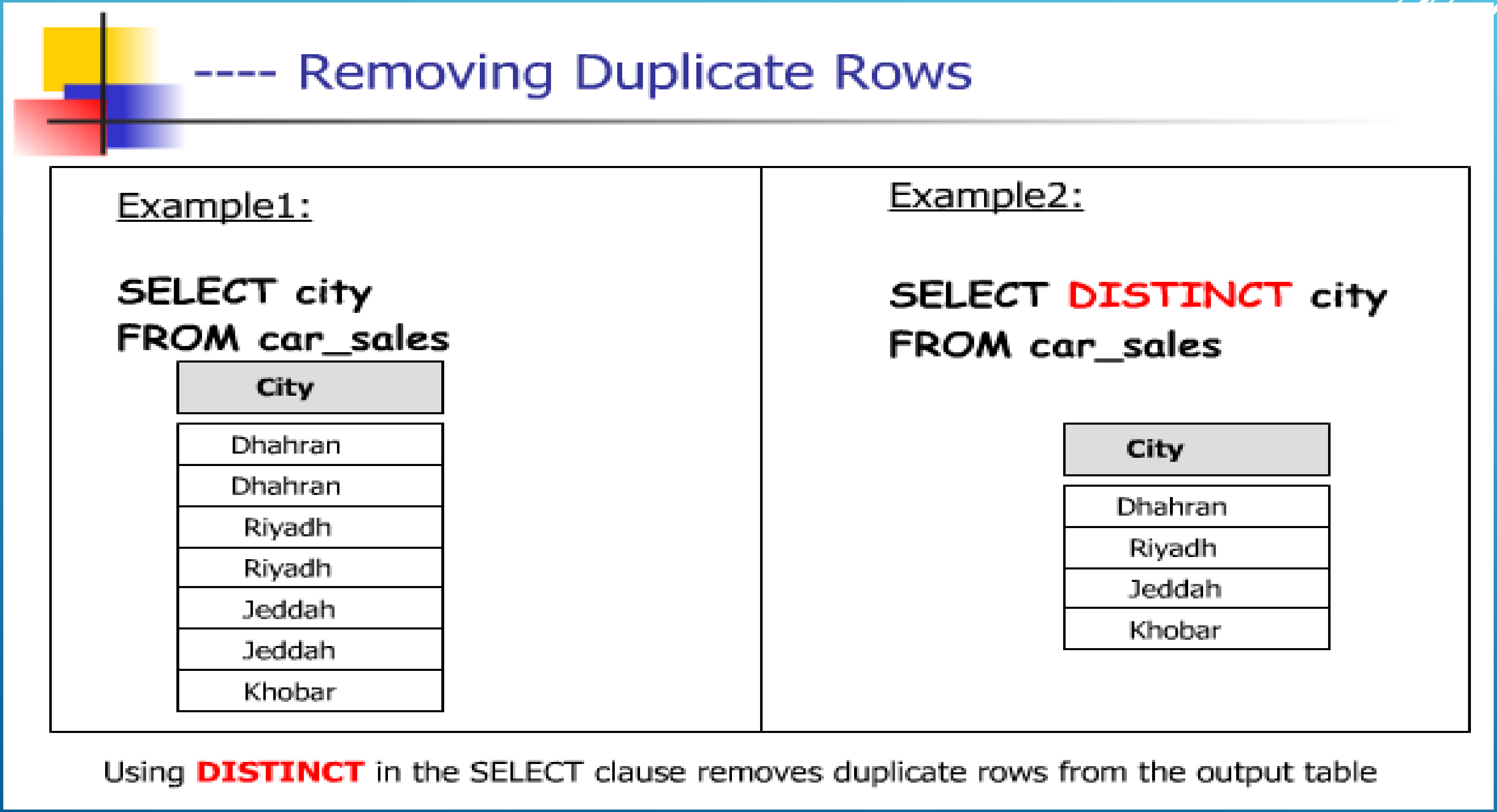
City
Khobar

Example 2: Select all cities where the number of cars sold **is kown.**

```
SELECT city
FROM car_sales
WHERE cars_sold IS NOT NULL;
```

City
Dhahran
Dhahran
Riyadh
Riyadh
Jeddah
Jeddah

SELECT Commands



---- Removing Duplicate Rows

Example1:

```
SELECT city
FROM car_sales
```

City
Dhahran
Dhahran
Riyadh
Riyadh
Jeddah
Jeddah
Khobar

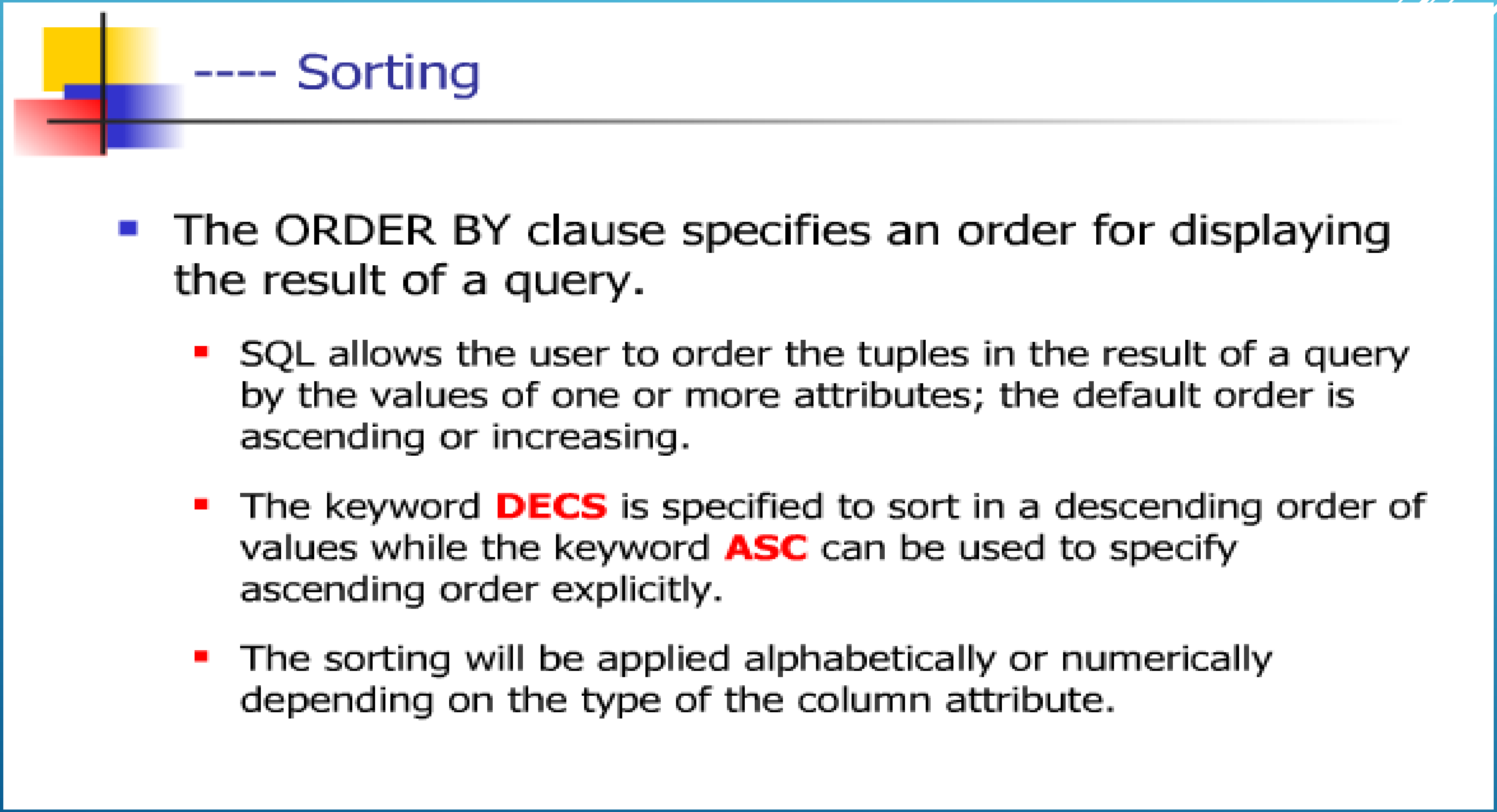
Example2:

```
SELECT DISTINCT city
FROM car_sales
```

City
Dhahran
Riyadh
Jeddah
Khobar

Using **DISTINCT** in the SELECT clause removes duplicate rows from the output table

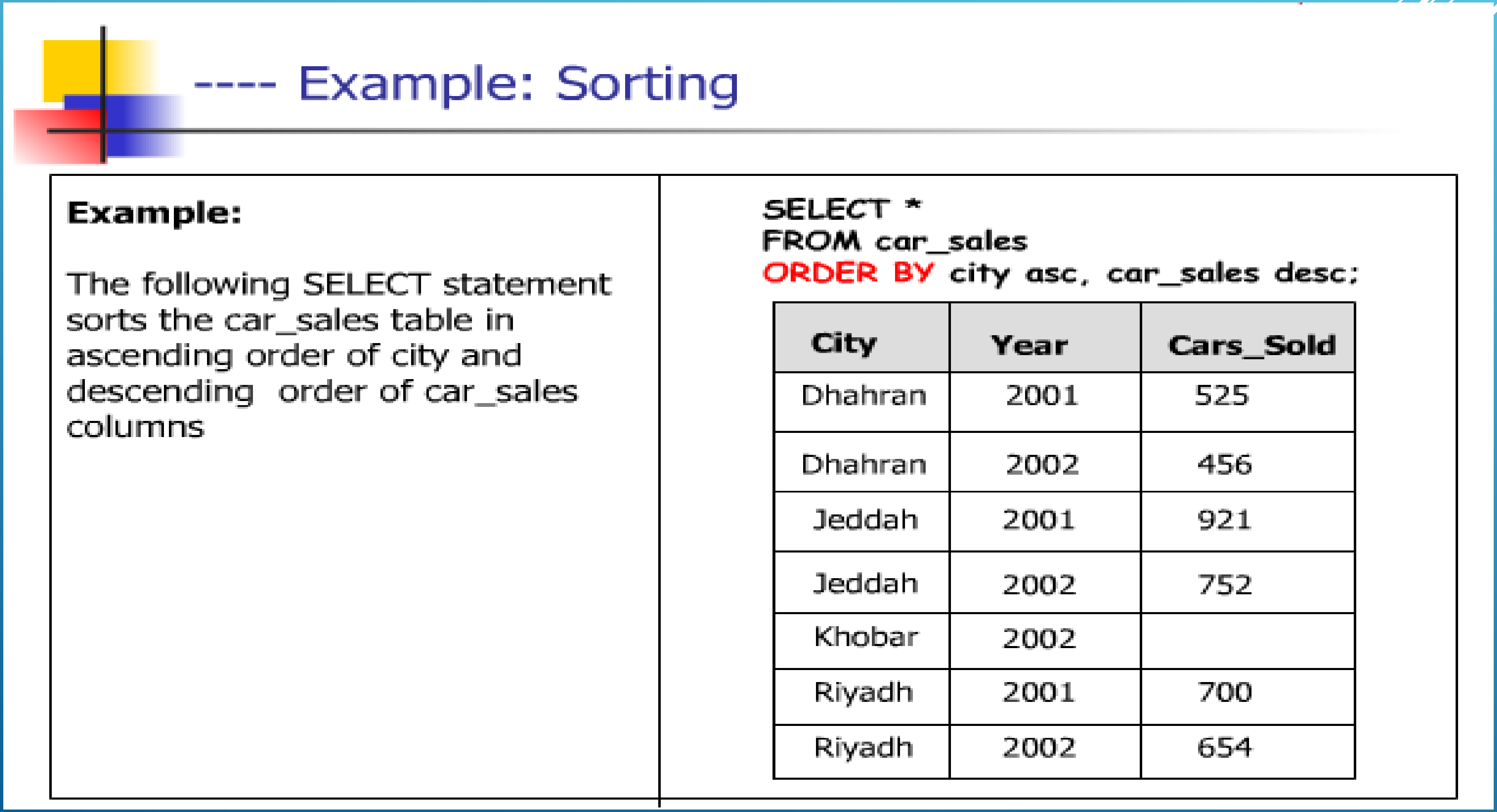
SELECT Commands



---- Sorting

- The ORDER BY clause specifies an order for displaying the result of a query.
  - SQL allows the user to order the tuples in the result of a query by the values of one or more attributes; the default order is ascending or increasing.
  - The keyword **DECS** is specified to sort in a descending order of values while the keyword **ASC** can be used to specify ascending order explicitly.
  - The sorting will be applied alphabetically or numerically depending on the type of the column attribute.

SELECT Commands



---- Example: Sorting

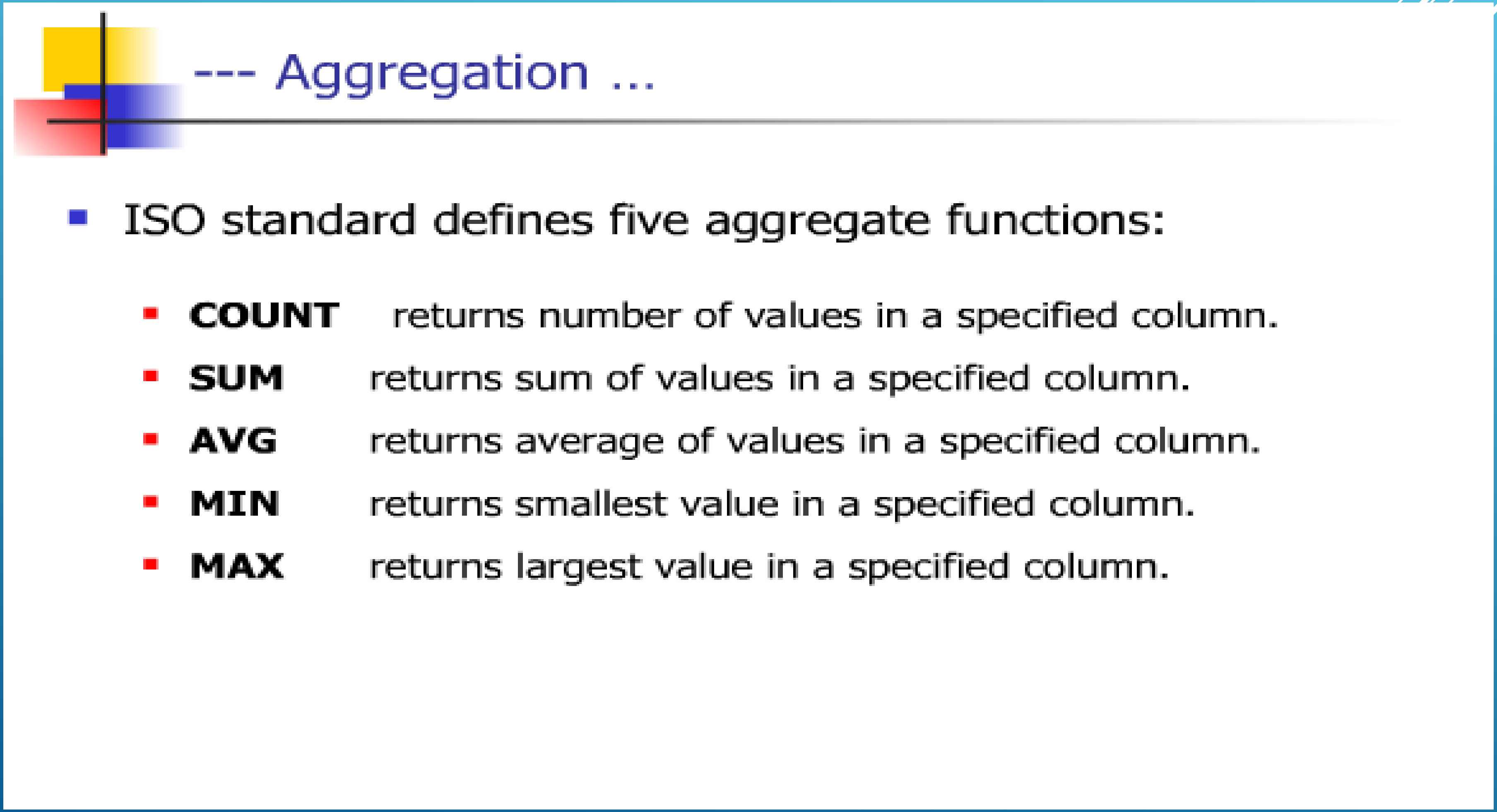
Example:

The following SELECT statement sorts the car_sales table in ascending order of city and descending order of car_sales columns

```
SELECT *
FROM car_sales
ORDER BY city asc, car_sales desc;
```

City	Year	Cars_Sold
Dhahran	2001	525
Dhahran	2002	456
Jeddah	2001	921
Jeddah	2002	752
Khobar	2002	
Riyadh	2001	700
Riyadh	2002	654

SELECT Commands

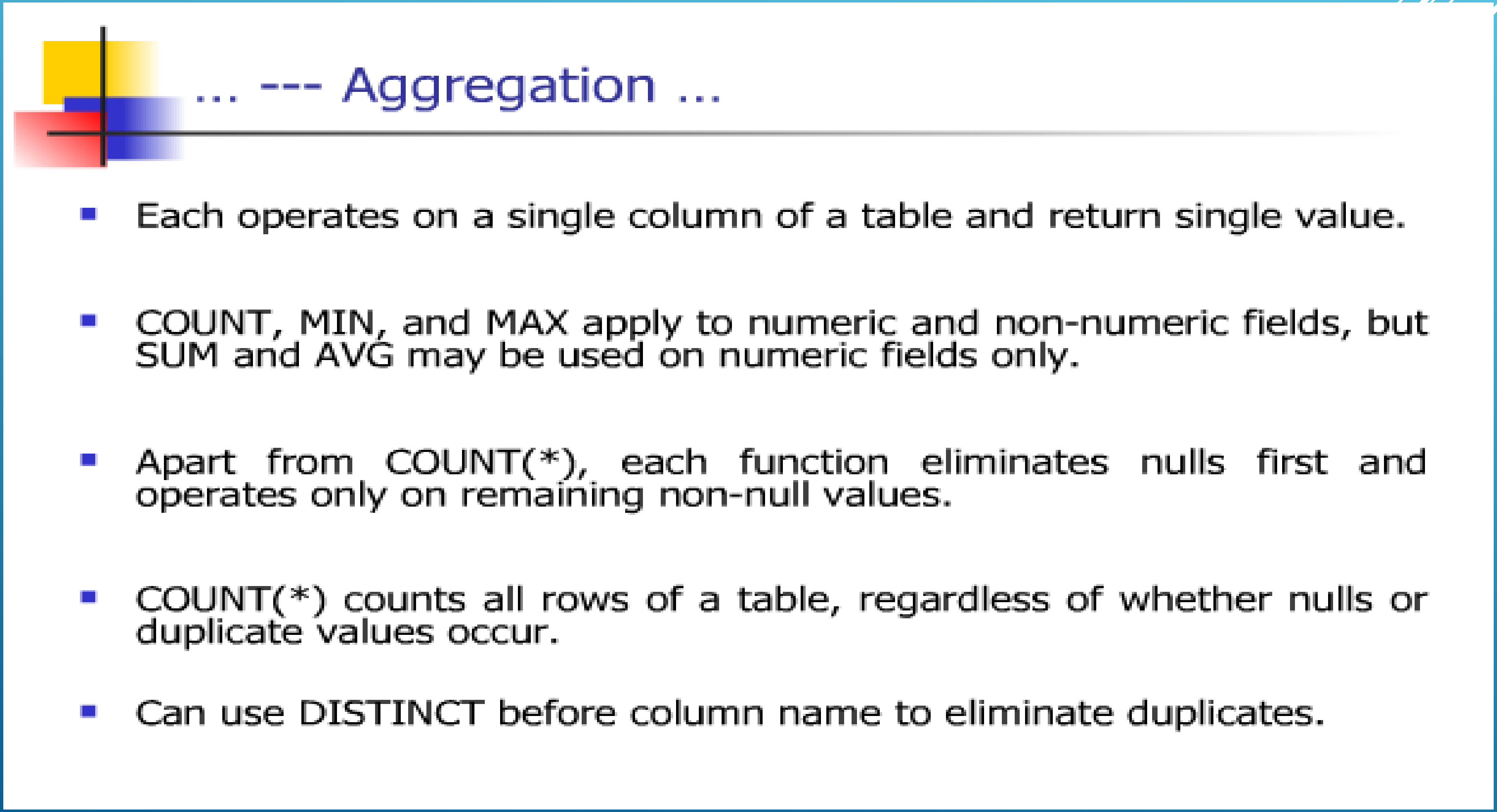


--- Aggregation ...

- ISO standard defines five aggregate functions:

COUNT	returns number of values in a specified column.
SUM	returns sum of values in a specified column.
AVG	returns average of values in a specified column.
MIN	returns smallest value in a specified column.
MAX	returns largest value in a specified column.

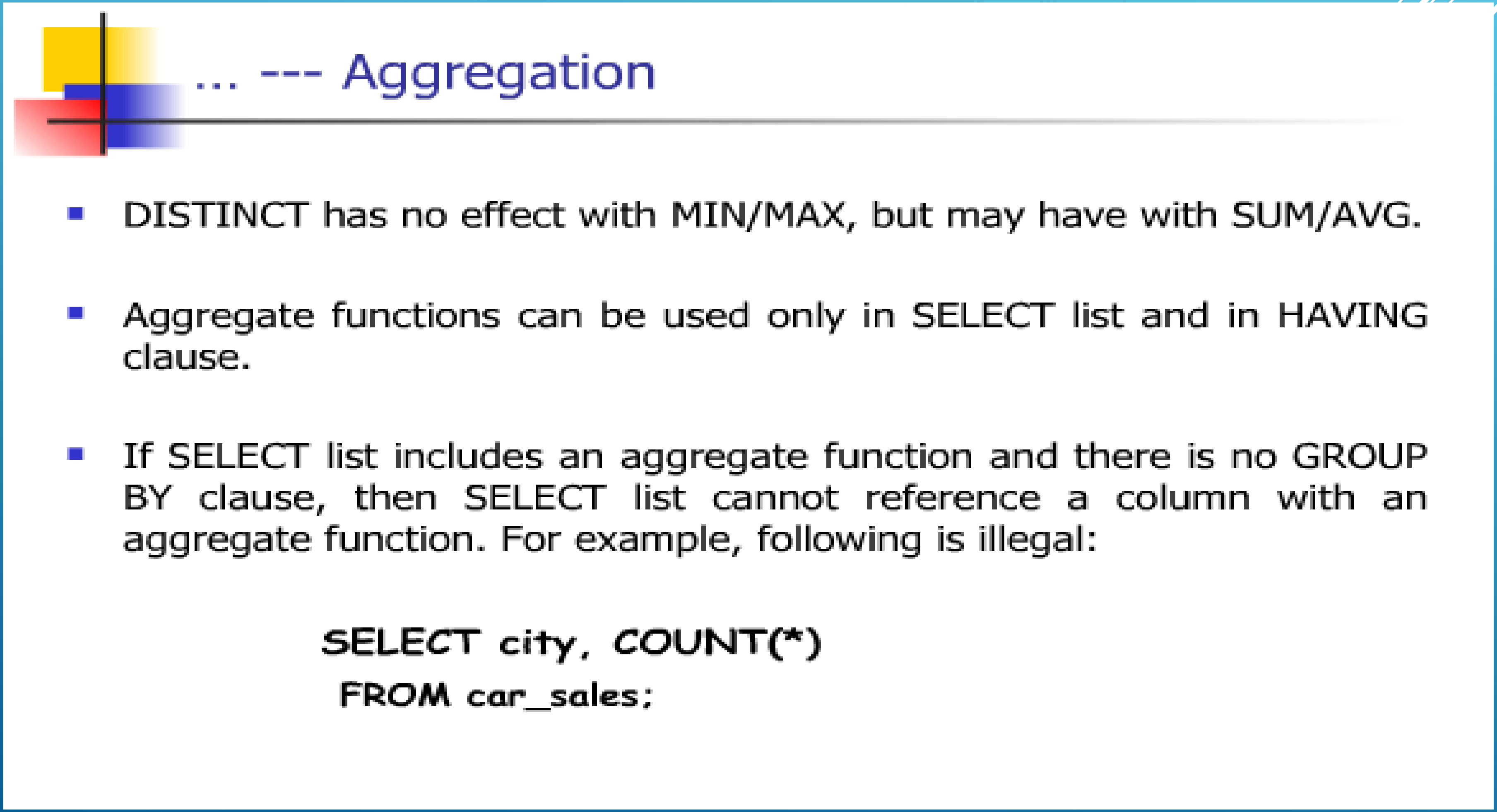
SELECT Commands



... --- Aggregation ...

- Each operates on a single column of a table and return single value.
- COUNT, MIN, and MAX apply to numeric and non-numeric fields, but SUM and AVG may be used on numeric fields only.
- Apart from COUNT(*), each function eliminates nulls first and operates only on remaining non-null values.
- COUNT(*) counts all rows of a table, regardless of whether nulls or duplicate values occur.
- Can use DISTINCT before column name to eliminate duplicates.

SELECT Commands

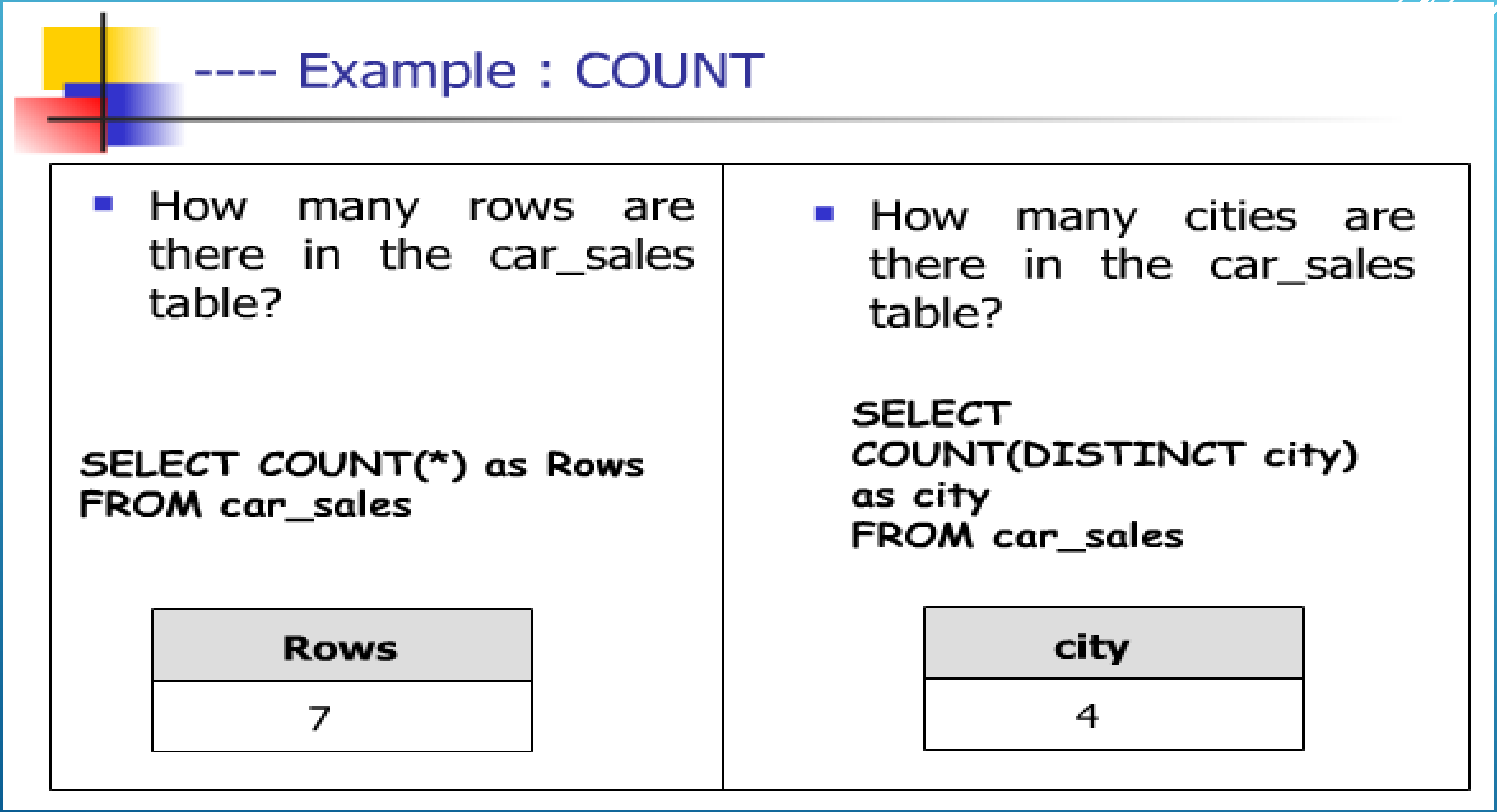


… --- Aggregation

- DISTINCT has no effect with MIN/MAX, but may have with SUM/AVG.
- Aggregate functions can be used only in SELECT list and in HAVING clause.
- If SELECT list includes an aggregate function and there is no GROUP BY clause, then SELECT list cannot reference a column with an aggregate function. For example, following is illegal:

```
SELECT city, COUNT(*)
 FROM car_sales;
```

SELECT Commands



---- Example : COUNT

- How many rows are there in the car_sales table?

```
SELECT COUNT(*) as Rows
FROM car_sales
```

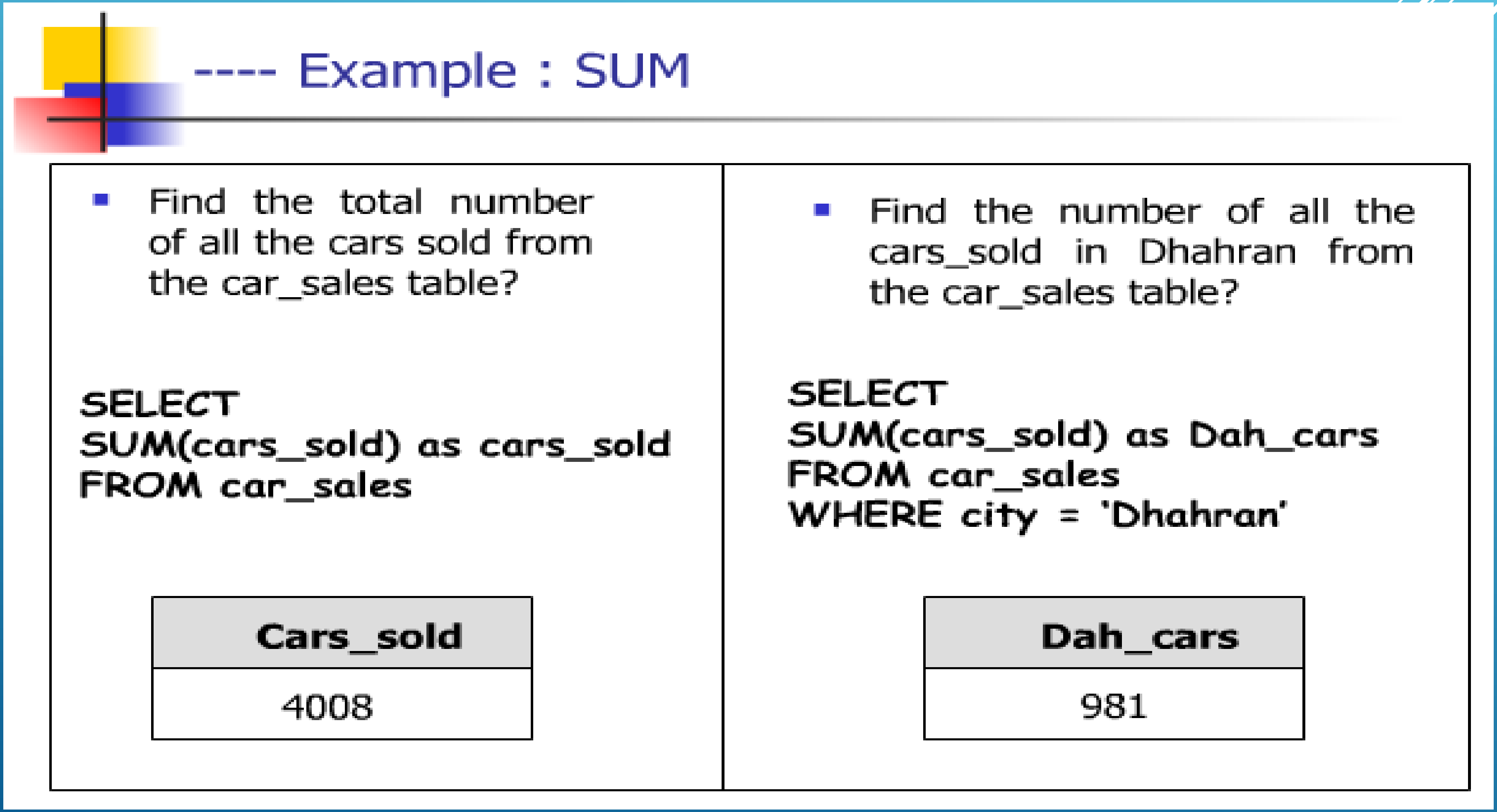
Rows
7

- How many cities are there in the car_sales table?

```
SELECT
COUNT(DISTINCT city)
as city
FROM car_sales
```

city
4

SELECT Commands



---- Example : SUM

▪ Find the total number of all the cars sold from the car_sales table?	▪ Find the number of all the cars_sold in Dhahran from the car_sales table?

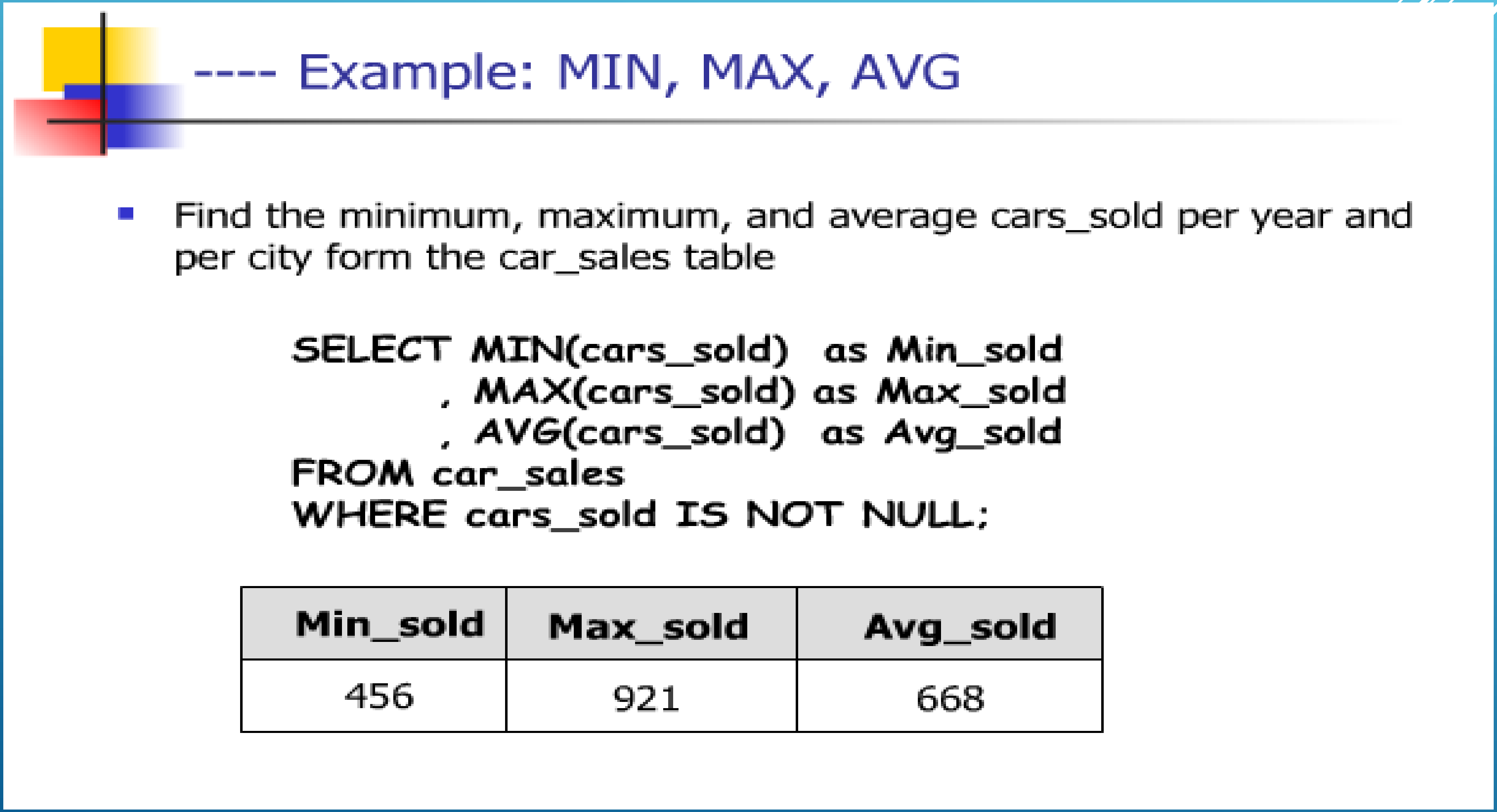
```
SELECT
SUM(cars_sold) as cars_sold
FROM car_sales
```

Cars_sold
4008

```
SELECT
SUM(cars_sold) as Dah_cars
FROM car_sales
WHERE city = 'Dhahran'
```

Dah_cars
981

SELECT Commands



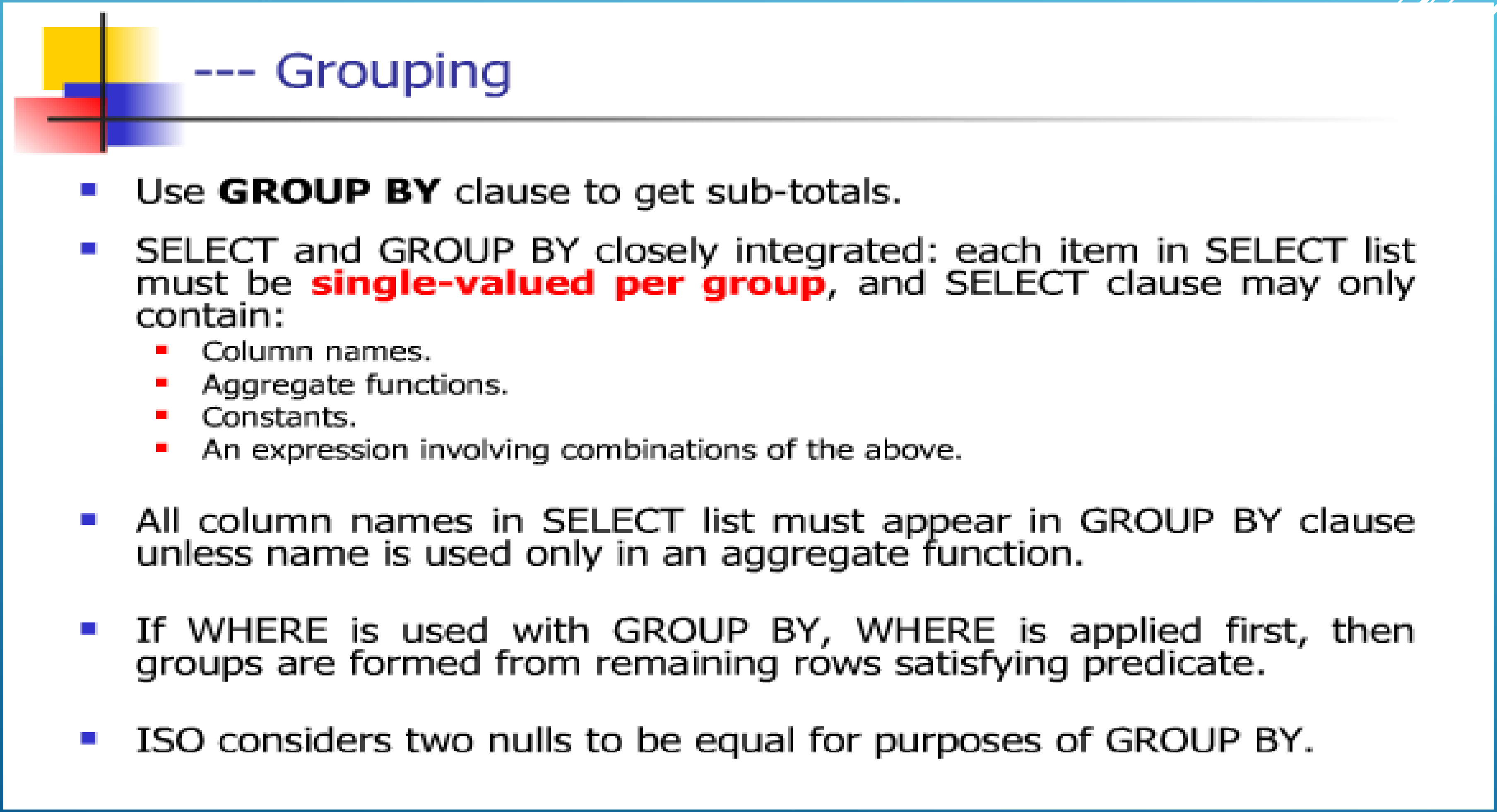
---- Example: MIN, MAX, AVG

- Find the minimum, maximum, and average cars_sold per year and per city form the car_sales table

```
SELECT MIN(cars_sold)  as Min_sold
     , MAX(cars_sold) as Max_sold
     , AVG(cars_sold)  as Avg_sold
FROM car_sales
WHERE cars_sold IS NOT NULL;
```

Min_sold	Max_sold	Avg_sold
456	921	668

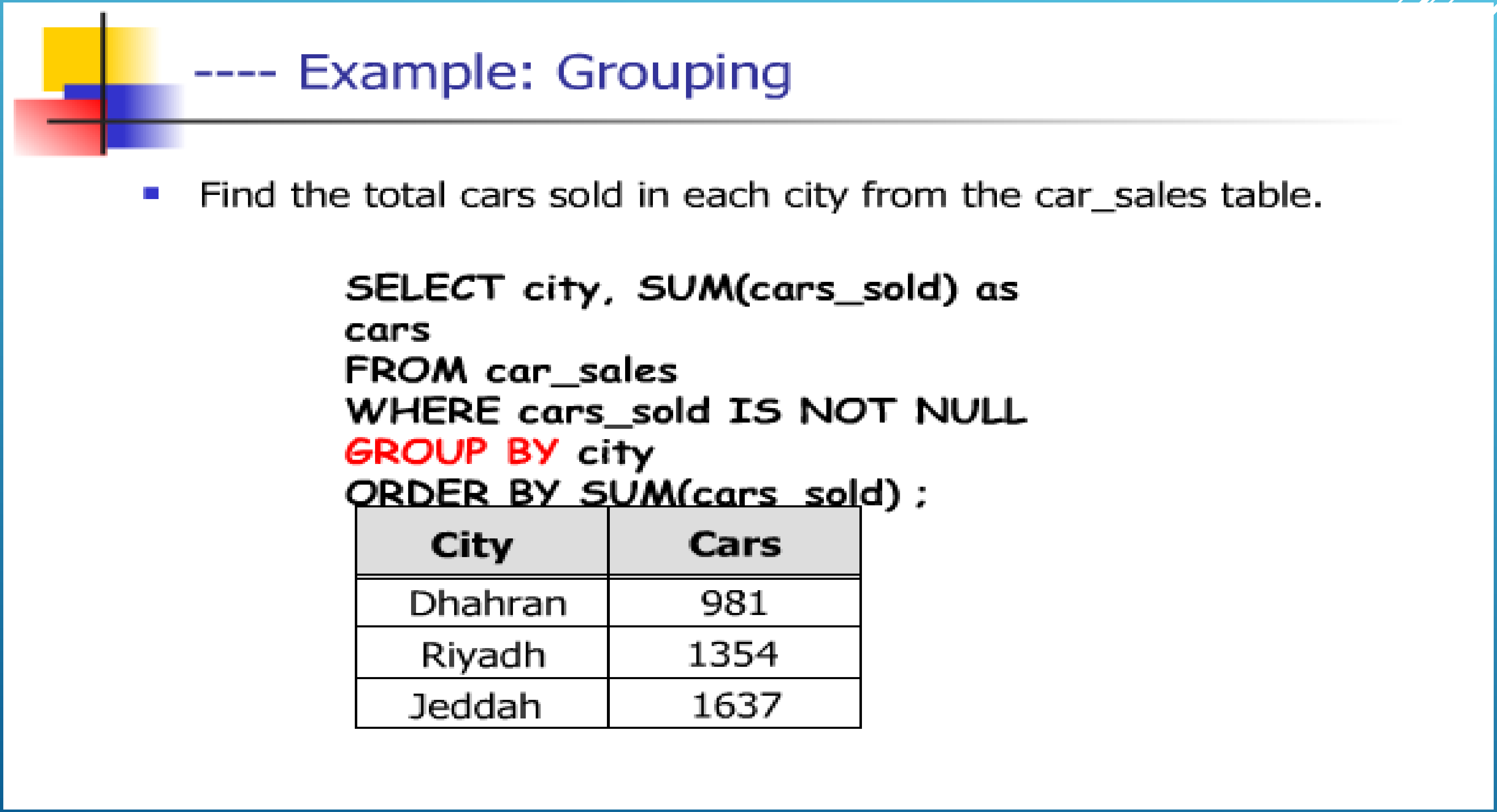
SELECT Commands



--- Grouping

- Use **GROUP BY** clause to get sub-totals.
- SELECT and GROUP BY closely integrated: each item in SELECT list must be **single-valued per group**, and SELECT clause may only contain:
  - Column names.
  - Aggregate functions.
  - Constants.
  - An expression involving combinations of the above.
- All column names in SELECT list must appear in GROUP BY clause unless name is used only in an aggregate function.
- If WHERE is used with GROUP BY, WHERE is applied first, then groups are formed from remaining rows satisfying predicate.
- ISO considers two nulls to be equal for purposes of GROUP BY.

SELECT Commands



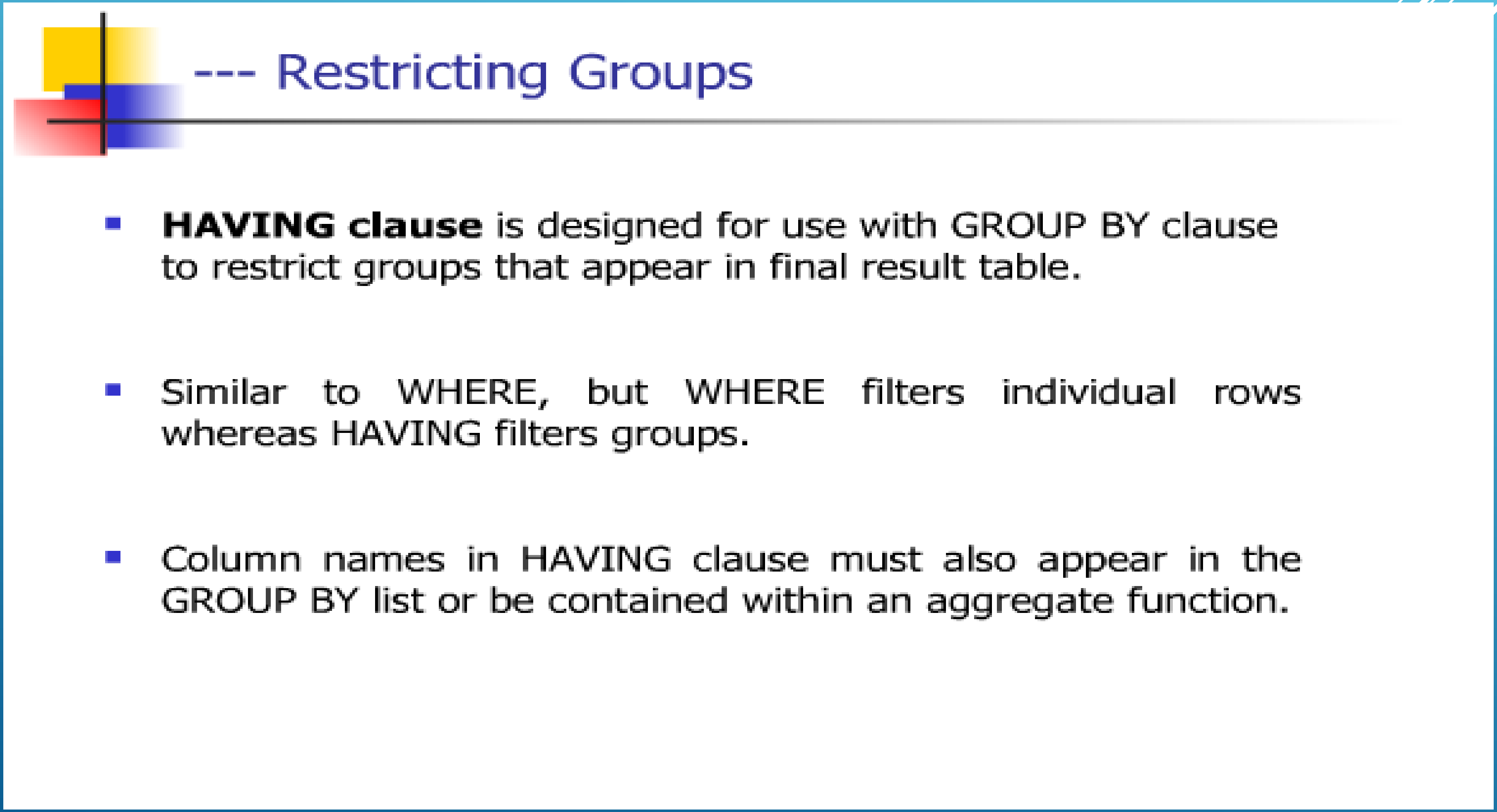
---- Example: Grouping

- Find the total cars sold in each city from the car_sales table.

```
SELECT city, SUM(cars_sold) as
cars
FROM car_sales
WHERE cars_sold IS NOT NULL
GROUP BY city
ORDER BY SUM(cars_sold) ;
```

City	Cars
Dhahran	981
Riyadh	1354
Jeddah	1637

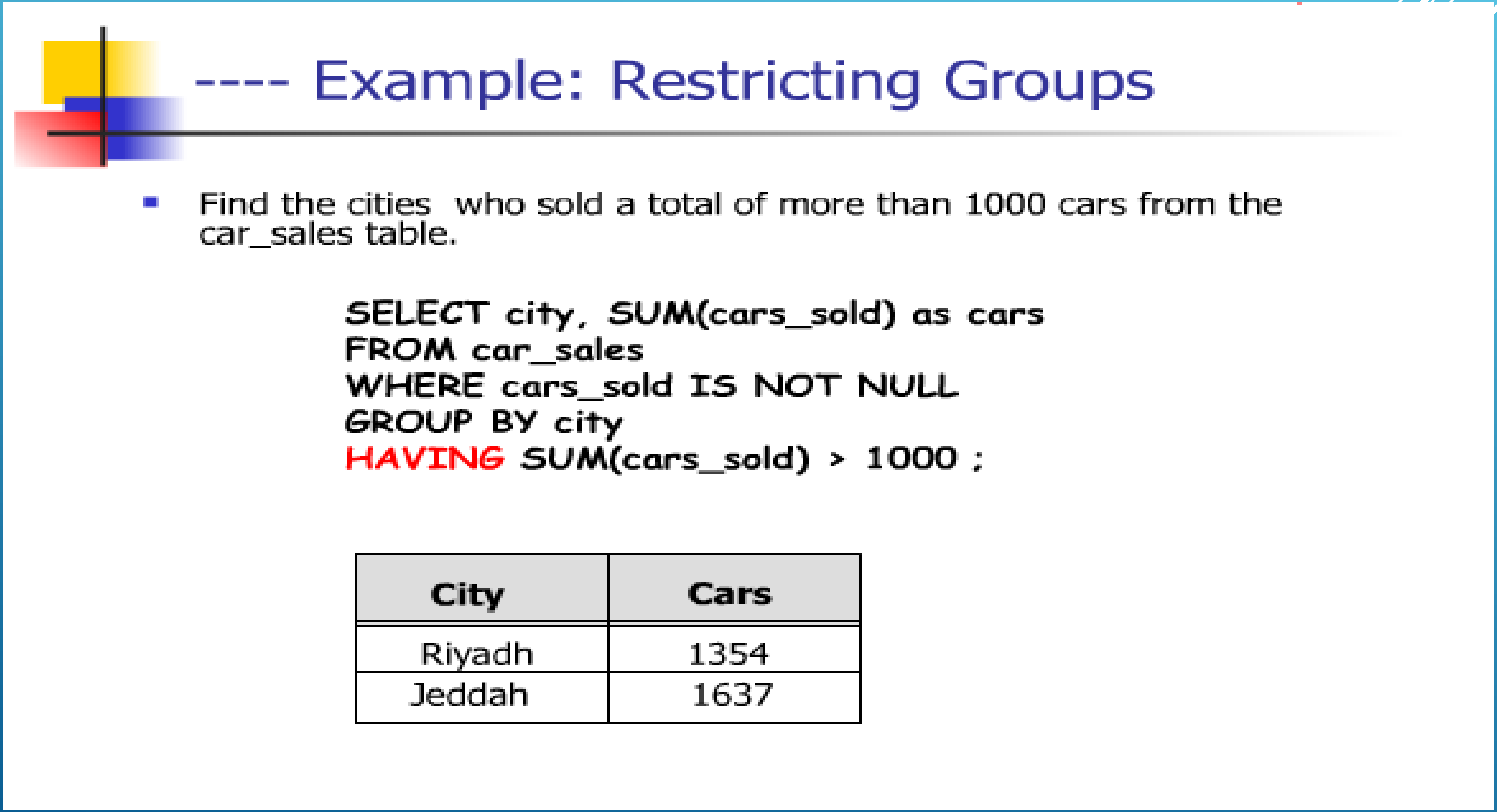
SELECT Commands



--- Restricting Groups

- **HAVING clause** is designed for use with GROUP BY clause to restrict groups that appear in final result table.
- Similar to WHERE, but WHERE filters individual rows whereas HAVING filters groups.
- Column names in HAVING clause must also appear in the GROUP BY list or be contained within an aggregate function.

SELECT Commands



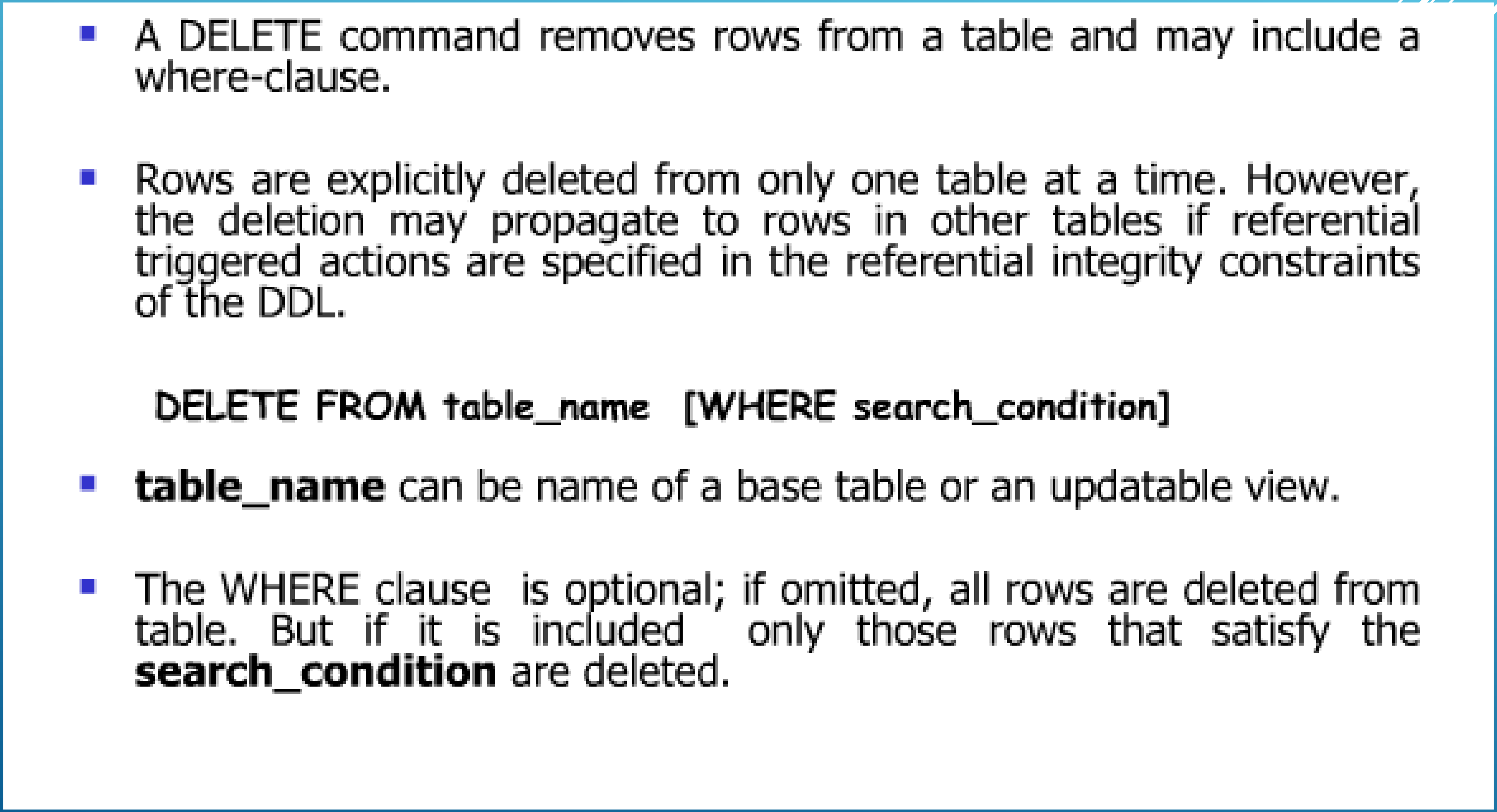
---- Example: Restricting Groups

- Find the cities who sold a total of more than 1000 cars from the car_sales table.

```
SELECT city, SUM(cars_sold) as cars
FROM car_sales
WHERE cars_sold IS NOT NULL
GROUP BY city
HAVING SUM(cars_sold) > 1000 ;
```

City	Cars
Riyadh	1354
Jeddah	1637

DELETE Commands



- A DELETE command removes rows from a table and may include a where-clause.
- Rows are explicitly deleted from only one table at a time. However, the deletion may propagate to rows in other tables if referential triggered actions are specified in the referential integrity constraints of the DDL.

```
DELETE FROM table_name   [WHERE search_condition]
```

- **table_name** can be name of a base table or an updatable view.
- The WHERE clause is optional; if omitted, all rows are deleted from table. But if it is included   only those rows that satisfy the **search_condition** are deleted.

DELETE Commands

- Delete all records from employee.

```
DELETE FROM employee;
```

- Delete all employees in department 1.

```
DELETE FROM employee
WHERE dno = 1;
```

TRUNCATE Commands

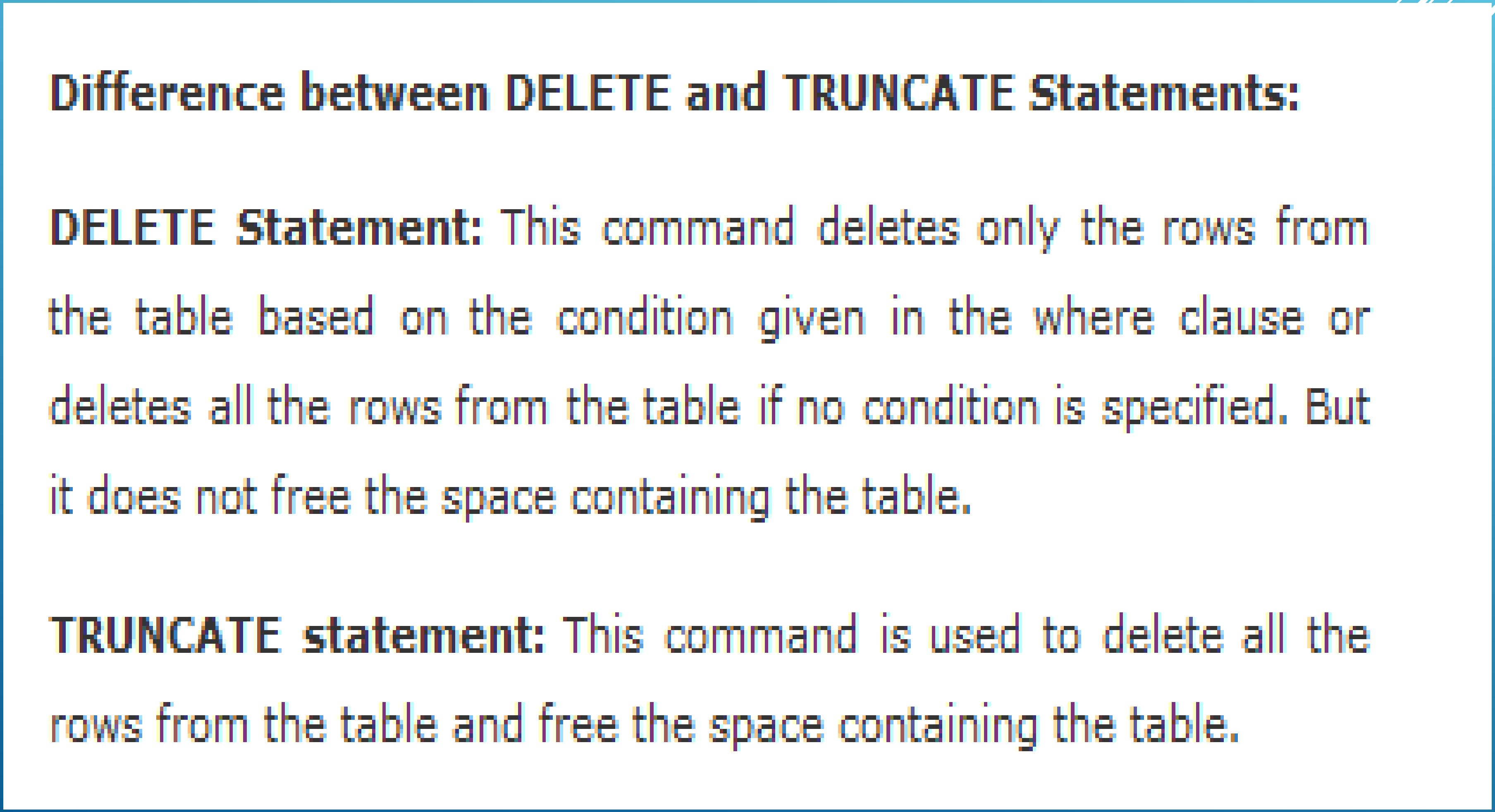
- Used to delete all the rows from the table and free the space containing the table.

```
TRUNCATE TABLE table_name;
```

eg: To delete all the rows from employee table

```
TRUNCATE TABLE employee;
```

TRUNCATE Commands

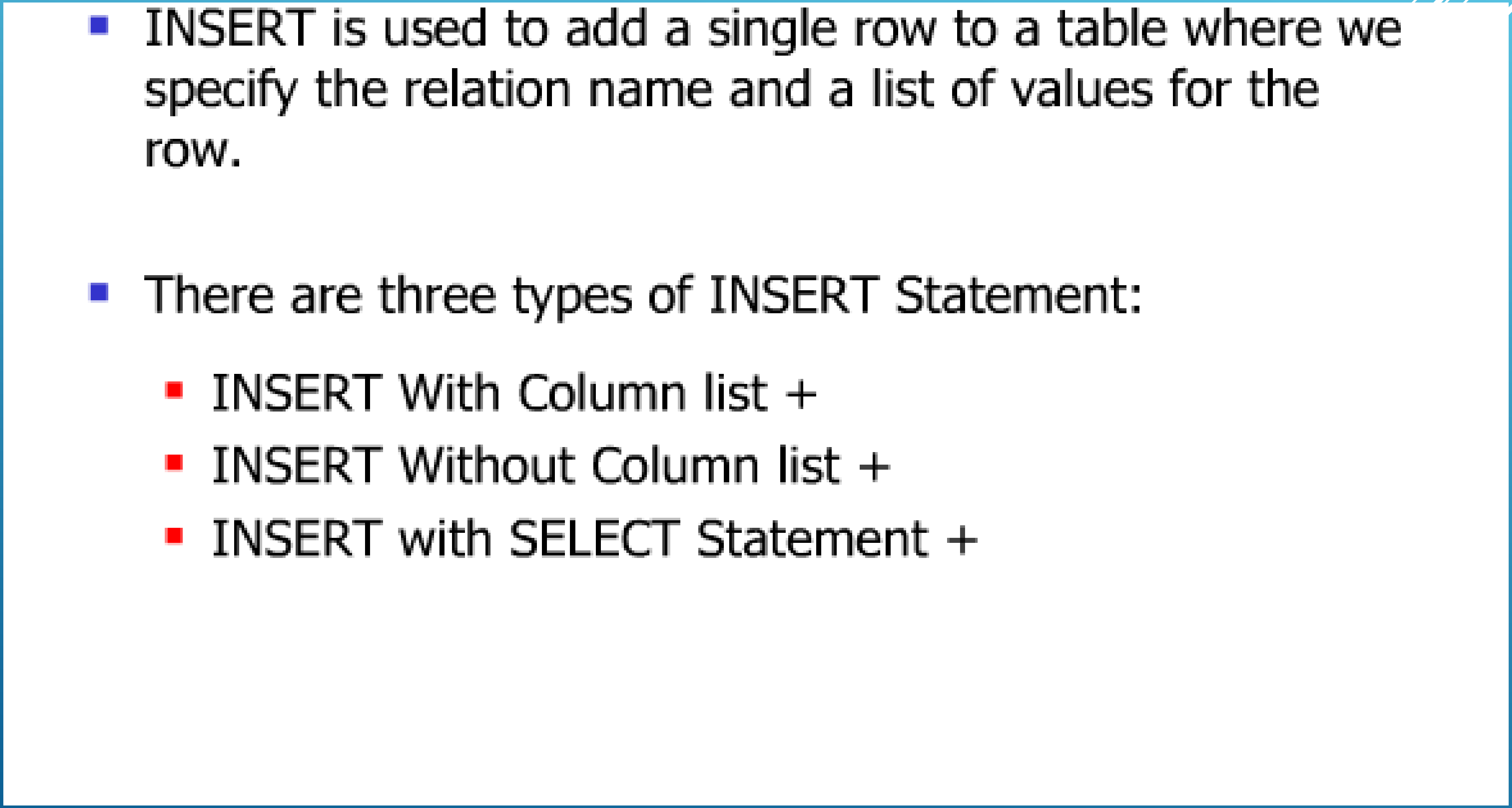


Difference between DELETE and TRUNCATE Statements:

DELETE Statement: This command deletes only the rows from the table based on the condition given in the where clause or deletes all the rows from the table if no condition is specified. But it does not free the space containing the table.

TRUNCATE statement: This command is used to delete all the rows from the table and free the space containing the table.

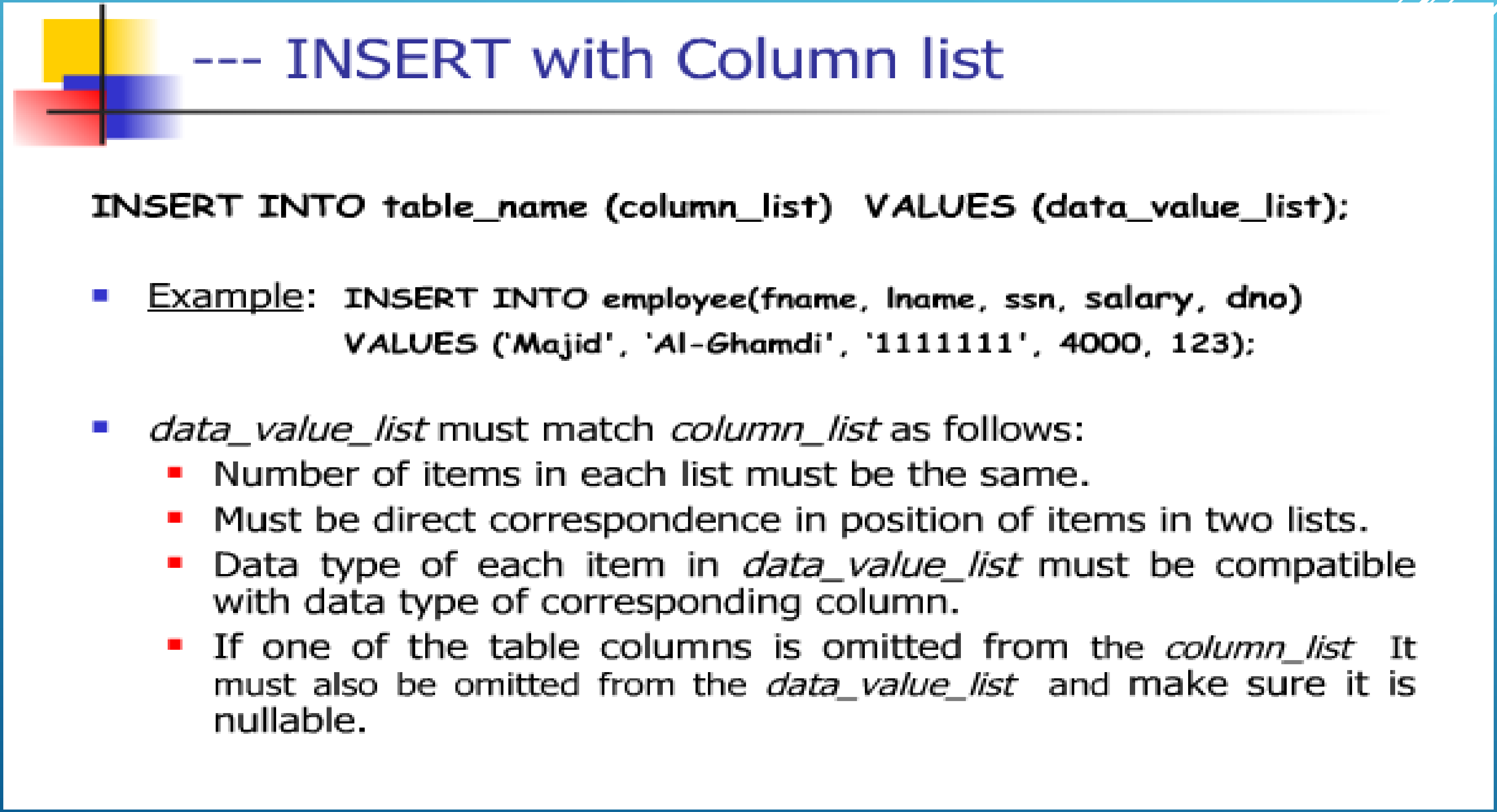
INSERT Commands



- INSERT is used to add a single row to a table where we specify the relation name and a list of values for the row.

- There are three types of INSERT Statement:
  - INSERT With Column list +
  - INSERT Without Column list +
  - INSERT with SELECT Statement +

INSERT Commands



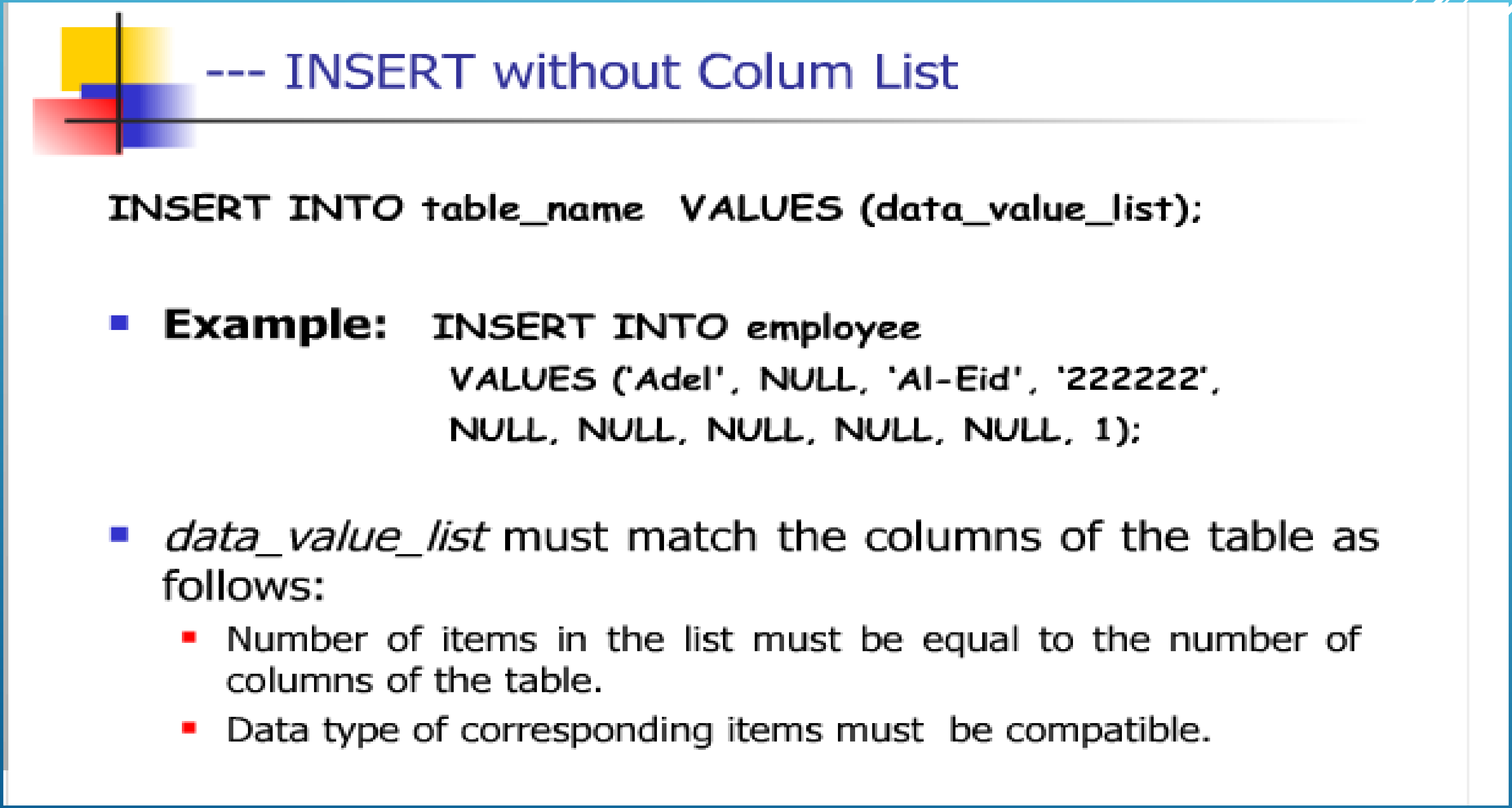
--- INSERT with Column list

```
INSERT INTO table_name (column_list)  VALUES (data_value_list);
```

- Example: `INSERT INTO employee(fname, lname, ssn, salary, dno)`
  `VALUES ('Majid', 'Al-Ghamdi', '1111111', 4000, 123);`

- *data_value_list* must match *column_list* as follows:
  - Number of items in each list must be the same.
  - Must be direct correspondence in position of items in two lists.
  - Data type of each item in *data_value_list* must be compatible with data type of corresponding column.
  - If one of the table columns is omitted from the *column_list* It must also be omitted from the *data_value_list* and make sure it is nullable.

INSERT Commands



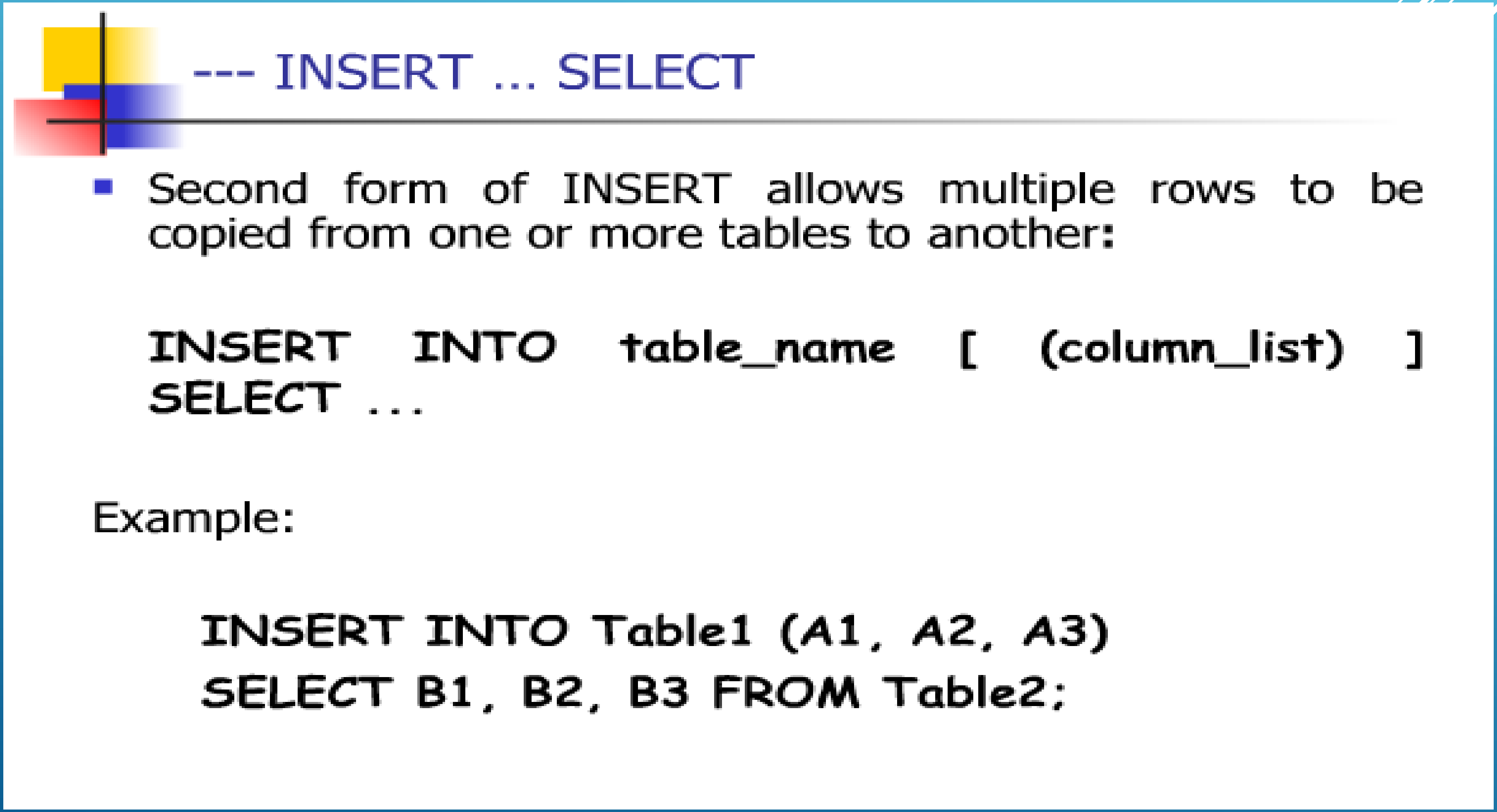
--- INSERT without Colum List

```
INSERT INTO table_name  VALUES (data_value_list);
```

- **Example:** `INSERT INTO employee`
  ```
  VALUES ('Adel', NULL, 'Al-Eid', '222222',
  NULL, NULL, NULL, NULL, NULL, 1);
  ```

- *data_value_list* must match the columns of the table as follows:
  - Number of items in the list must be equal to the number of columns of the table.
  - Data type of corresponding items must be compatible.

INSERT Commands



--- INSERT ... SELECT

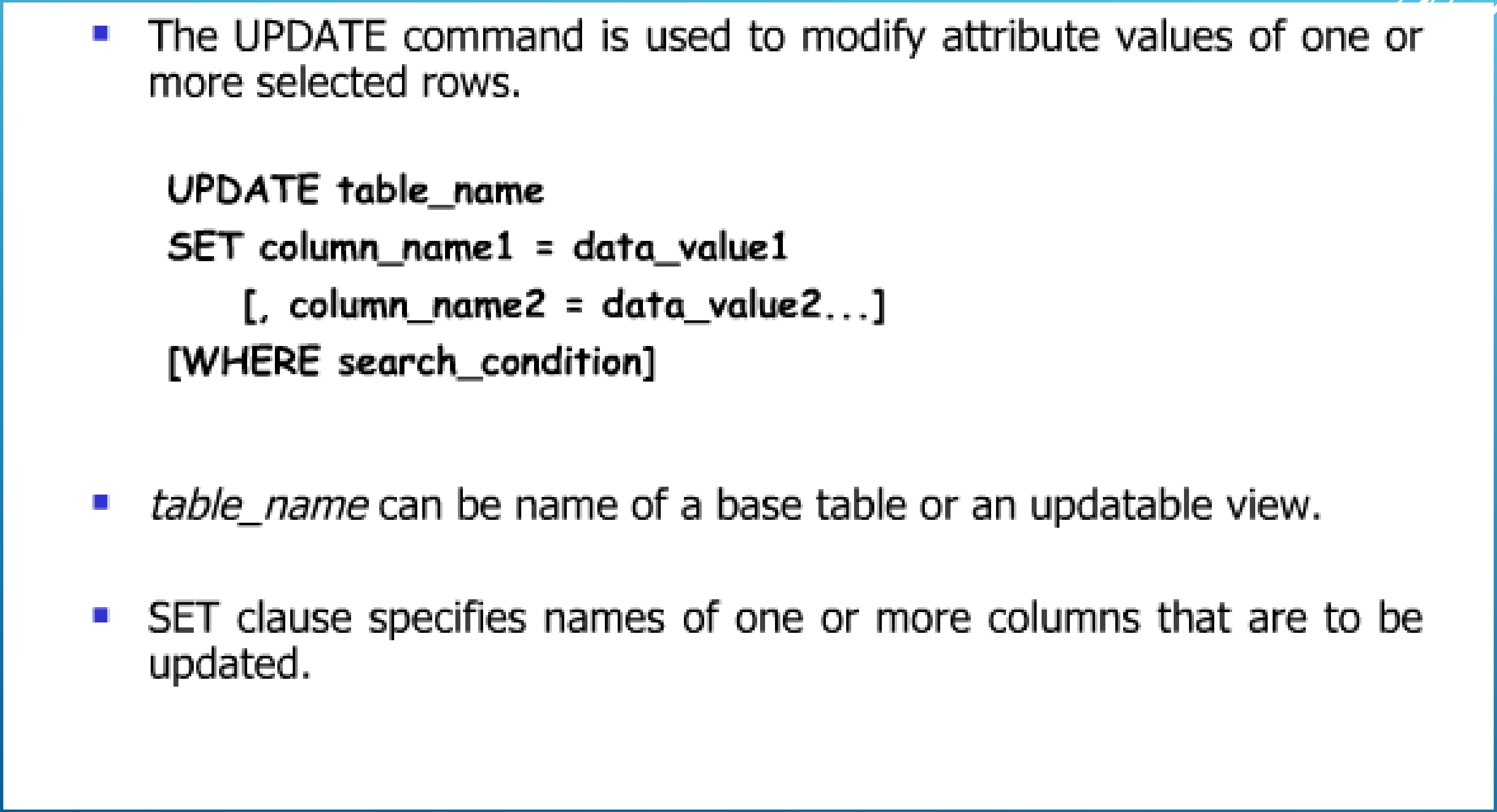
- Second form of INSERT allows multiple rows to be copied from one or more tables to another:

```
INSERT INTO table_name [ (column_list) ]
SELECT ...
```

Example:

```
INSERT INTO Table1 (A1, A2, A3)
SELECT B1, B2, B3 FROM Table2;
```

UPDATE Commands



- The UPDATE command is used to modify attribute values of one or more selected rows.

```
UPDATE table_name
SET column_name1 = data_value1
     [, column_name2 = data_value2...]
[WHERE search_condition]
```

- *table_name* can be name of a base table or an updatable view.

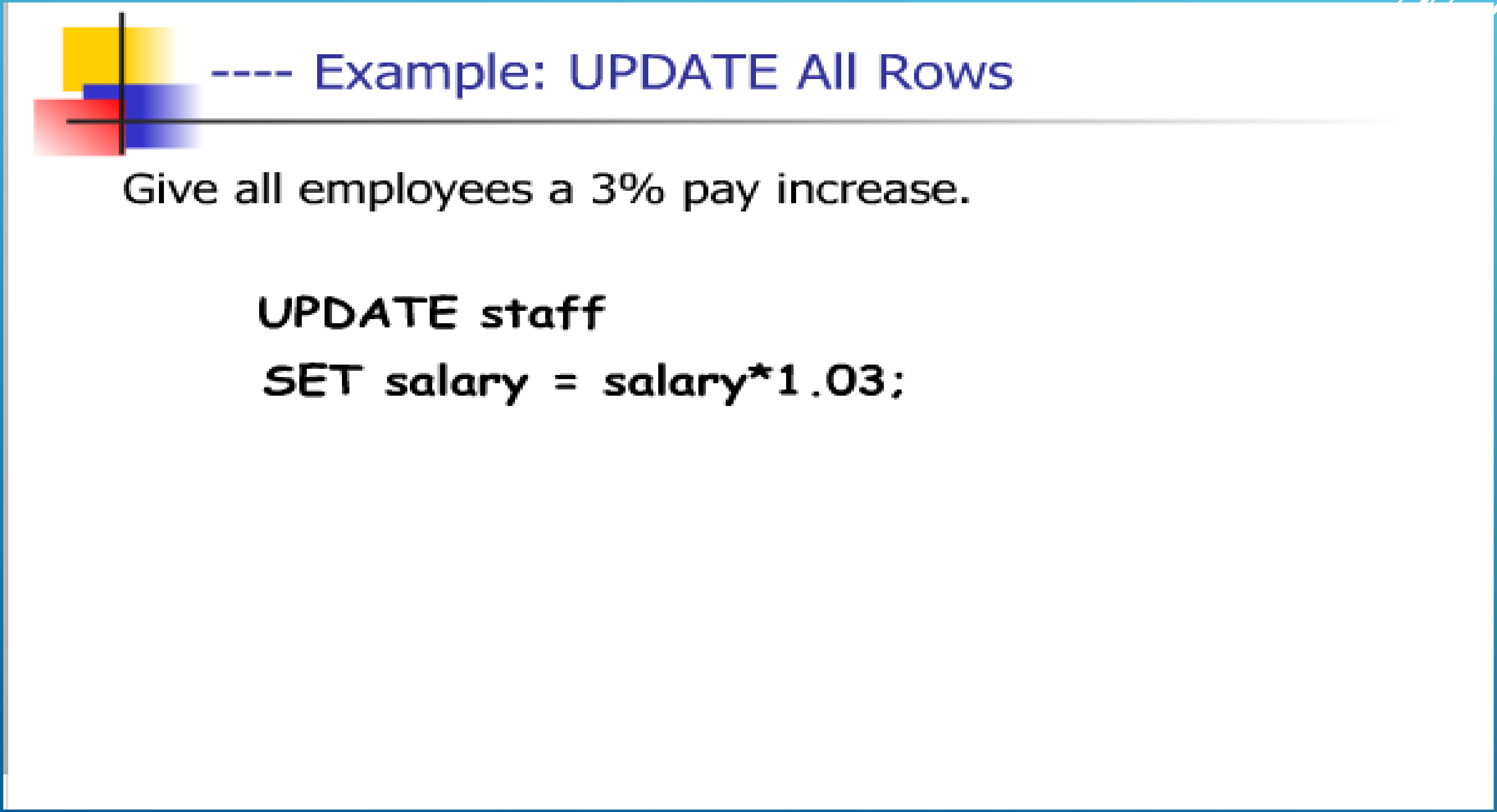
- SET clause specifies names of one or more columns that are to be updated.

UPDATE Commands

- WHERE clause is optional:
  - If omitted, named columns are updated for all rows in table.
  - If specified, only those rows that satisfy *search_condition* are updated.

- New *data_value(s)* must be compatible with data type for corresponding column.

UPDATE Commands

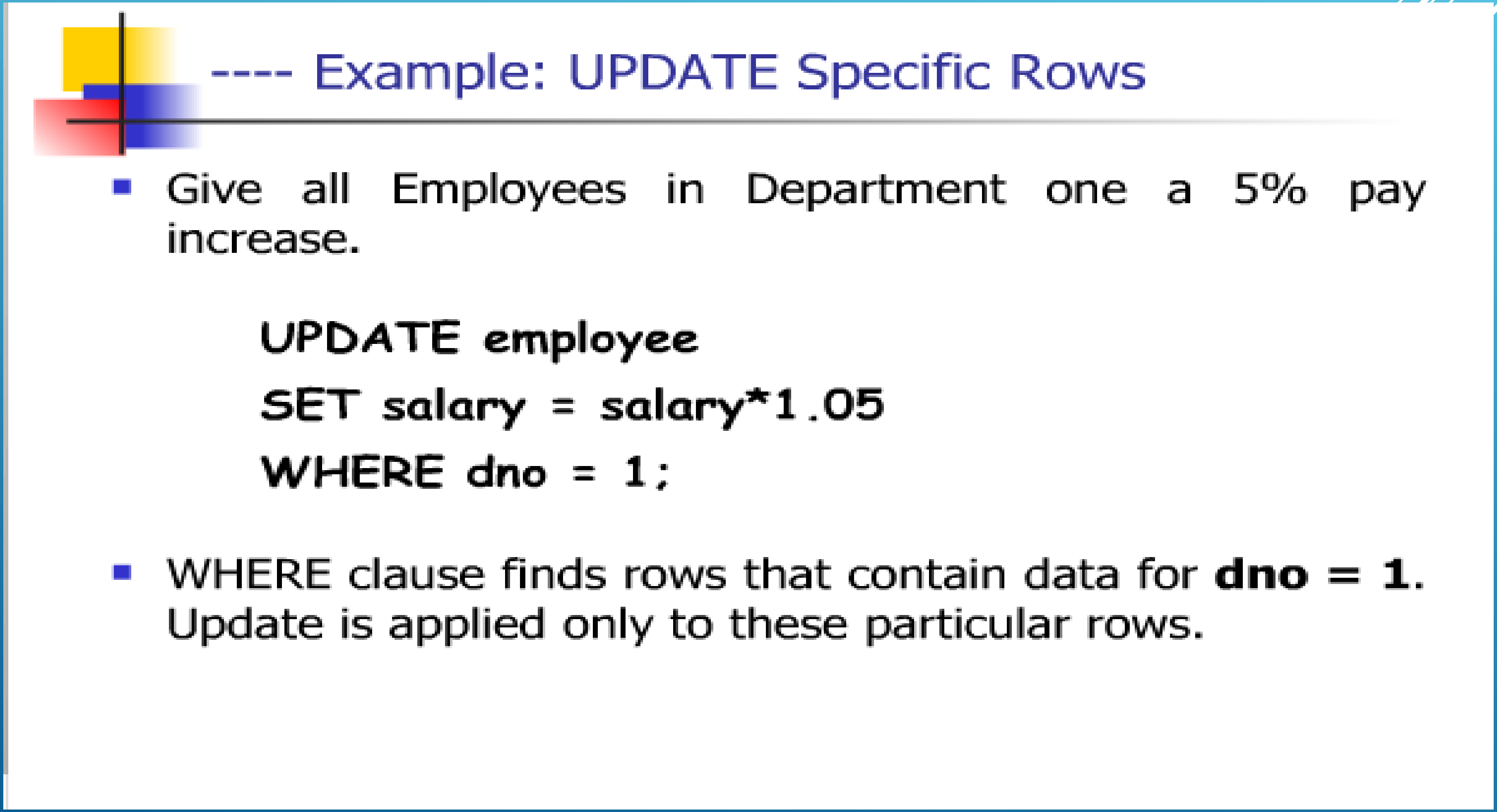


---- Example: UPDATE All Rows

Give all employees a 3% pay increase.

```
UPDATE staff
SET salary = salary*1.03;
```

UPDATE Commands



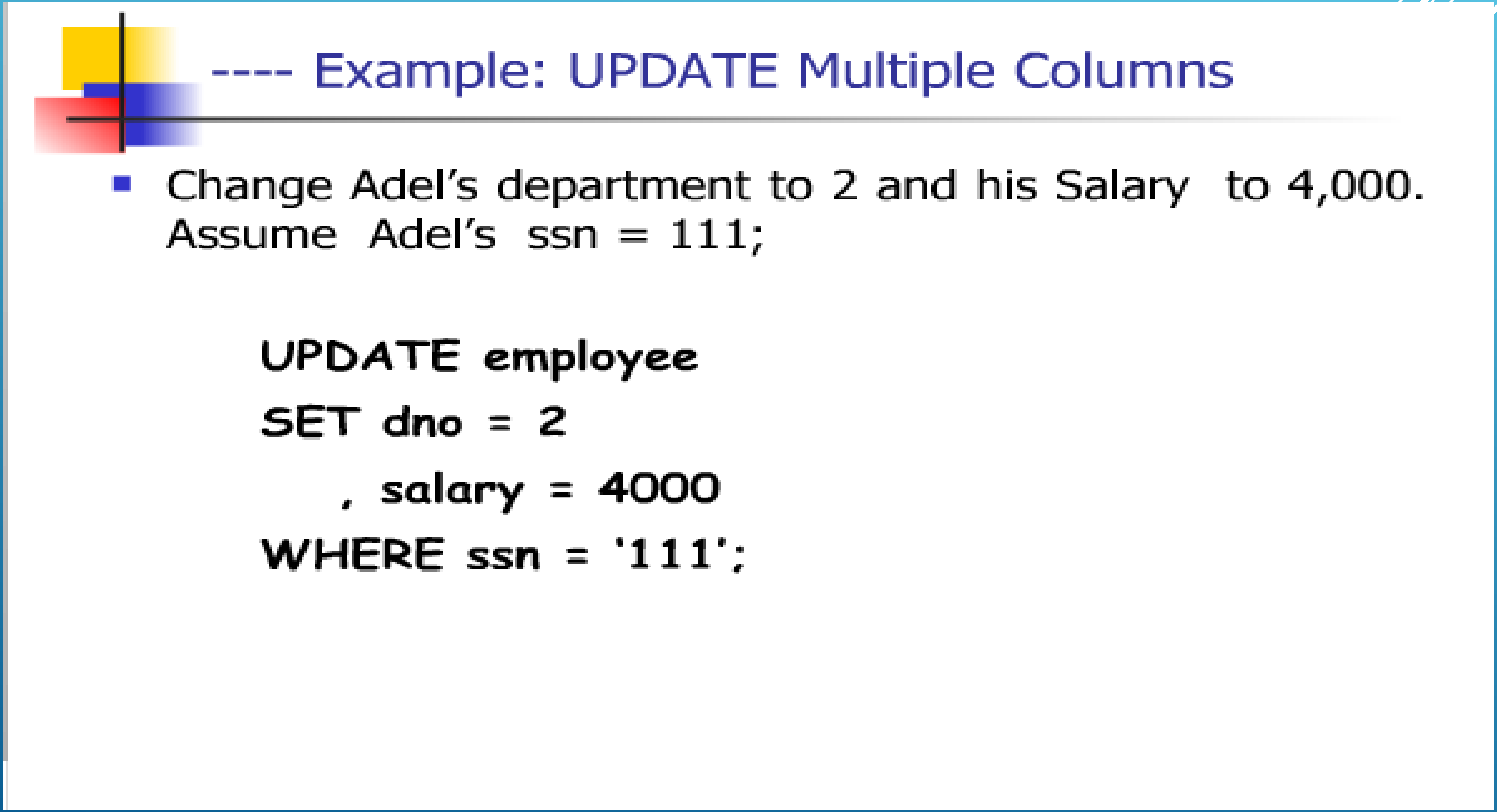
---- Example: UPDATE Specific Rows

- Give all Employees in Department one a 5% pay increase.

```
UPDATE employee
SET salary = salary*1.05
WHERE dno = 1;
```

- WHERE clause finds rows that contain data for **dno = 1**. Update is applied only to these particular rows.

UPDATE Commands

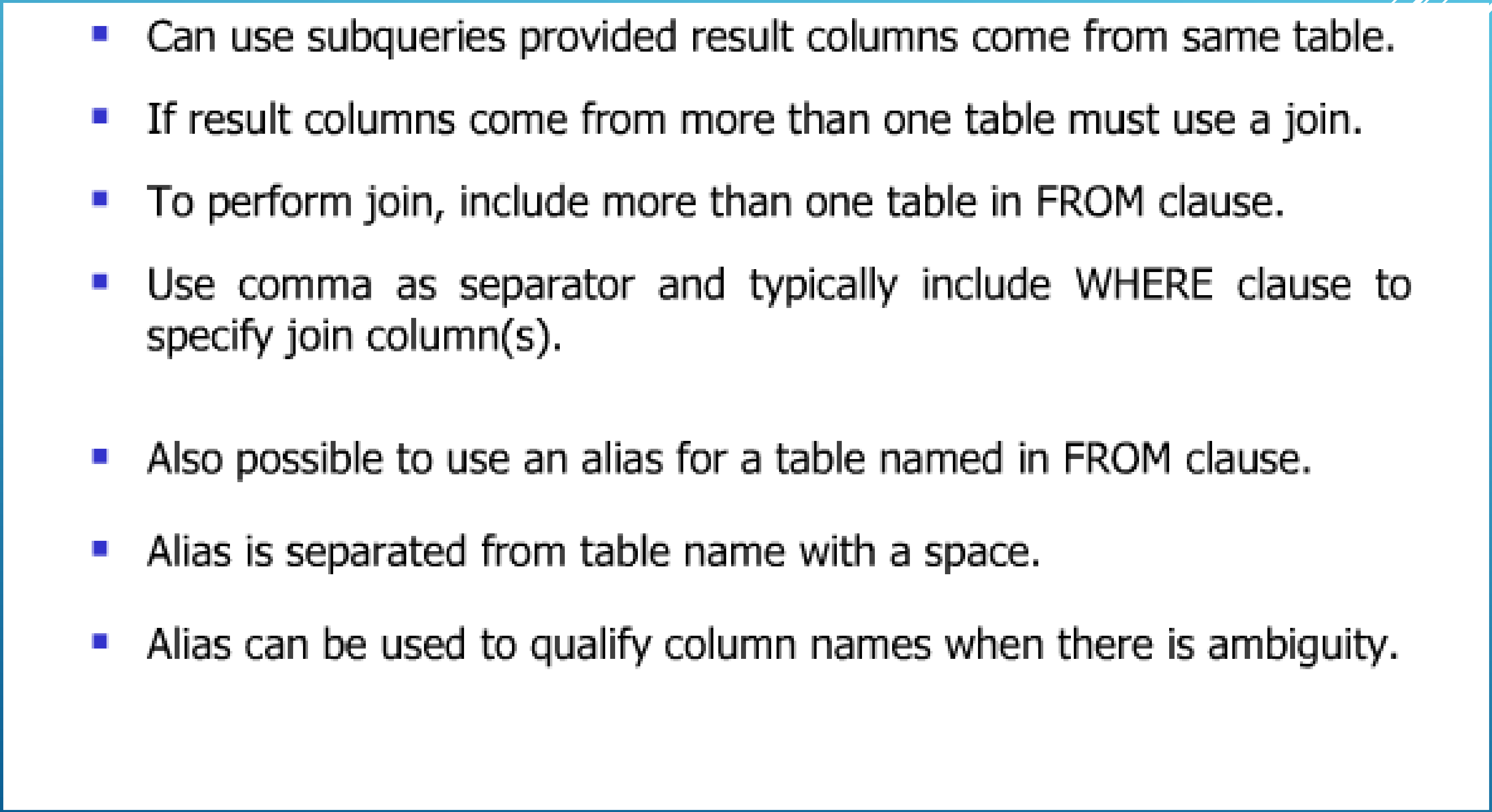


---- Example: UPDATE Multiple Columns

- Change Adel's department to 2 and his Salary to 4,000. Assume Adel's ssn = 111;

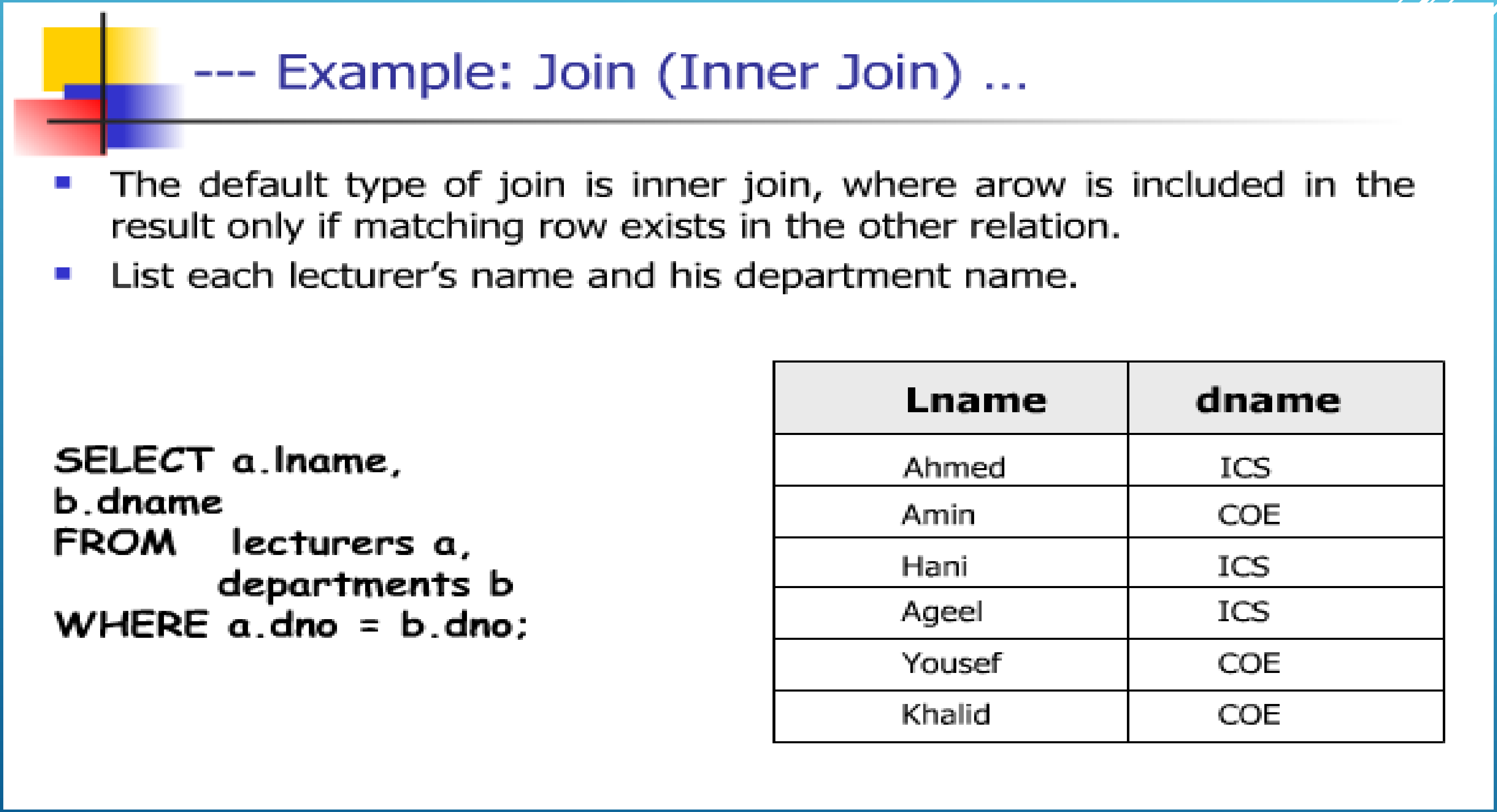
```
UPDATE employee
SET dno = 2
    , salary = 4000
WHERE ssn = '111';
```

JOIN Commands



- Can use subqueries provided result columns come from same table.
- If result columns come from more than one table must use a join.
- To perform join, include more than one table in FROM clause.
- Use comma as separator and typically include WHERE clause to specify join column(s).
- Also possible to use an alias for a table named in FROM clause.
- Alias is separated from table name with a space.
- Alias can be used to qualify column names when there is ambiguity.

JOIN Commands



--- Example: Join (Inner Join) ...

- The default type of join is inner join, where arow is included in the result only if matching row exists in the other relation.
- List each lecturer's name and his department name.

```
SELECT a.lname,
b.dname
FROM    lecturers a,
        departments b
WHERE a.dno = b.dno;
```

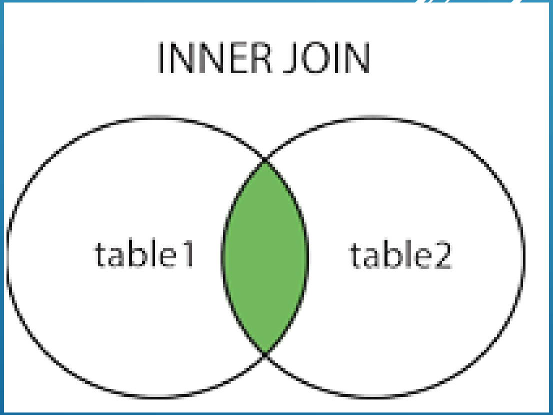
Lname	dname
Ahmed	ICS
Amin	COE
Hani	ICS
Ageel	ICS
Yousef	COE
Khalid	COE

SQL INNER JOIN Keyword

The INNER JOIN keyword selects records that have matching values in both tables.

INNER JOIN Syntax

```
SELECT column_name(s)
FROM table1
INNER JOIN table2
ON table1.column_name = table2.column_name;
```

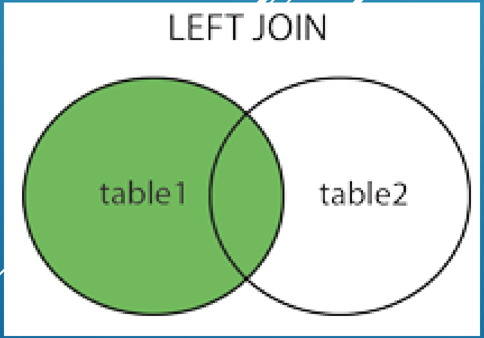


SQL LEFT JOIN Keyword

The LEFT JOIN keyword returns all records from the left table (table1), and the matching records from the right table (table2). The result is 0 records from the right side, if there is no match.

LEFT JOIN Syntax

```
SELECT column_name(s)
FROM table1
LEFT JOIN table2
ON table1.column_name = table2.column_name;
```

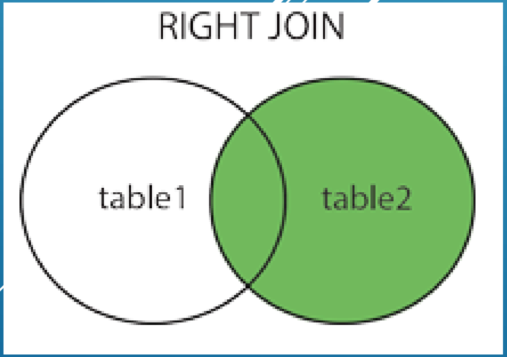


SQL RIGHT JOIN Keyword

The RIGHT JOIN keyword returns all records from the right table (table2), and the matching records from the left table (table1). The result is 0 records from the left side, if there is no match.

RIGHT JOIN Syntax

```
SELECT column_name(s)
FROM table1
RIGHT JOIN table2
ON table1.column_name = table2.column_name;
```

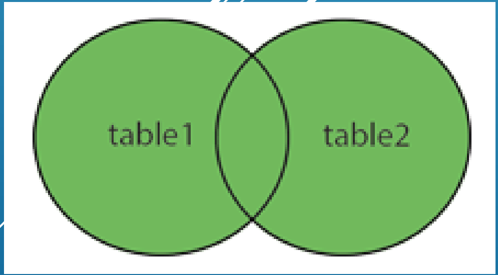


SQL FULL OUTER JOIN Keyword

The FULL OUTER JOIN keyword returns all records when there is a match in left (table1) or right (table2) table records.

FULL OUTER JOIN Syntax

```
SELECT column_name(s)
FROM table1
FULL OUTER JOIN table2
ON table1.column_name = table2.column_name
WHERE condition;
```



SQL COMMIT AND ROLLBACK

COMMIT and **ROLLBACK** are performed on transactions. A transaction is the smallest unit of work that is performed against a database. Its a sequence of instructions in a logical order. A transaction can be performed manually by a programmer or it can be triggered using an automated program.

SQL COMMIT

COMMIT is the SQL command that is used for storing changes performed by a transaction. When a **COMMIT** command is issued it saves all the changes since last **COMMIT** or **ROLLBACK.**

Syntax

```
COMMIT;
```

```
DELETE from Customer where State = 'Texas';
COMMIT;
```

SQL ROLLBACK

ROLLBACK is the SQL command that is used for reverting changes performed by a transaction. When a **ROLLBACK** command is issued it reverts all the changes since last **COMMIT** or **ROLLBACK**. The database returns to the state without any of the previous changes made by activity of the transaction

Syntax

```
ROLLBACK;
```

```
DELETE from Customer where State = 'Texas';
ROLLBACK;
```

THANK YOU