BCNF (Boyce-Codd Normal Form) is a higher level of database normalization used to eliminate redundancy and ensure data integrity. It is a stricter version of the Third Normal Form (3NF).

BCNF Definition

A table is in **BCNF** if:

1. It is already in **3NF**.
2. For every **functional dependency** ($X \rightarrow Y$), **X** should be a **superkey** (i.e., X should be either a primary key or contain the primary key).

Steps to Convert to BCNF

1. Identify all **functional dependencies** (FDs).
2. Check if the left-hand side (LHS) of every FD is a **superkey**.
3. If any FD violates BCNF (i.e., LHS is not a superkey), decompose the table into smaller tables while preserving dependencies.

Example: Breaking into BCNF

Given Table:

BCNF Definition

A table is in **BCNF** if:

1. It is already in **3NF**.
2. For every **functional dependency** ($X \rightarrow Y$), **X** should be a **superkey** (i.e., X should be either a primary key or contain the primary key).

Steps to Convert to BCNF

1. Identify all **functional dependencies** (FDs).
2. Check if the left-hand side (LHS) of every FD is a **superkey**.
3. If any FD violates BCNF (i.e., LHS is not a superkey), decompose the table into smaller tables while preserving dependencies.

Example: Breaking into BCNF

Given Table:

Student_ID	Course	Instructor
1	Math	Prof. A
2	Science	Prof. B
3	Math	Prof. A
4	Science	Prof. B

BCNF (Boyce-Codd Normal Form) is a higher level of database normalization used to eliminate redundancy and ensure data integrity. It is a stricter version of the Third Normal Form (3NF).

BCNF Definition

A table is in **BCNF** if:

1. It is already in **3NF**.
2. For every **functional dependency** ($X \rightarrow Y$), **X** should be a **superkey** (i.e., X should be either a primary key or contain the primary key).

Steps to Convert to BCNF

1. Identify all **functional dependencies** (FDs).
2. Check if the left-hand side (LHS) of every FD is a **superkey**.
3. If any FD violates BCNF (i.e., LHS is not a superkey), decompose the table into smaller tables while preserving dependencies.

Example: Breaking into BCNF

Given Table:

Student_ID	Course	Instructor
1	Math	Prof. A
2	Science	Prof. B

DML Statements

SQL WHERE clause

- To select specific rows from a table, you use a WHERE clause in the SELECT statement.
- The WHERE clause contains one or more logical expressions that evaluate each row in the table.
- If a row that causes the condition evaluates to true, it will be included in the result set; otherwise, it will be excluded.

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Comparison operators

Operator	Meaning
=	Equal to

- The WHERE clause contains one or more logical expressions that evaluate each row in the table.
- If a row that causes the condition evaluates to true, it will be included in the result set; otherwise, it will be excluded.

```
SELECT column1, column2, ...
FROM table_name
WHERE condition;
```

Comparison operators

Operator	Meaning
=	Equal to
<> (!=)	Not equal to
<	Less than
>	Greater than
<=	Less than or equal
>=	Greater than or equal

SQL WHERE clause Logical Operators

- A logical operator allows you to test for the truth of a condition.

SQL WHERE clause Logical Operators

- A logical operator allows you to test for the truth of a condition.

Logical Operators

S.No	Operator	Meaning
1	ALL	Return true if all comparisons are true
2	AND	Return true if both expressions are true
3	ANY	Return true if any one of the comparisons is true.
4	BETWEEN	Return true if the operand is within a range
5	EXISTS	Return true if a subquery contains any rows
6	IN	Return true if the operand is equal to one of the value in a list
7	LIKE	Return true if the operand matches a pattern
8	NOT	Reverse the result of any other Boolean operator.
9	OR	Return true if either expression is true
10	SOME	Return true if some of the expressions are true

SQL WHERE clause

Example

- *SQL WHERE clause with numeric comparison*

```
SELECT employee_id, first_name, last_name, salary
FROM [dbo].[Employees]
WHERE salary > 14000
```

employee_id	first_name	last_name	salary
100	Steven	King	24000.00
101	Neena	Kochhar	00.00
102	Lex	De Haan	17000.00

employees

- employee_id
- first_name
- last_name
- email
- phone_number
- hire_date
- job_id
- salary
- manager_id
- department_id

- *SQL WHERE clause with characters comparison*

```
SELECT employee_id, first_name, last_name
FROM [dbo].[Employees]
WHERE last_name = 'Chen';
```

employee_id	first_name	last_name
110	John	Chen

- *SQL WHERE clause with date comparison*

```
SELECT employee_id, first_name, last_name, hire_date
FROM [dbo].[Employees]
WHERE hire_date >= '1999-01-01';
```

employee_id	first_name	last_name	hire_date
179	Charles	Johnson	2000-01-04
113	Luis	Popp	1999-12-07
119	Karen	Colmenares	1999-08-10
178	Kimberely	Grant	1999-05-24
107	Diana	Lorentz	1999-02-07

Note: Use YEAR function to get the year from hire_date column
WHERE YEAR (hire_date) = 1999

SQL WHERE clause Logical Operators

- The LIKE operator compares a value to similar values using a wildcard operator.
  - The percent sign (%) represents zero, one, or multiple characters.
  - The underscore sign (_) represents a single character.

Example

- *SQL WHERE clause with LIKE operator*

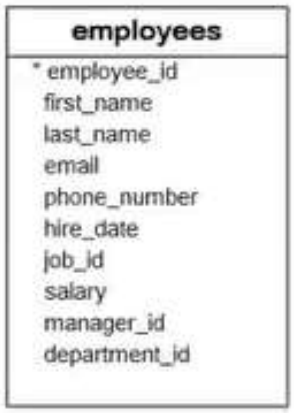
```
SELECT employee_id, first_name, last_name
FROM [dbo].[Employees]
WHERE first_name LIKE 'jo%';
```

employee_id	first_name	last_name
110	John	Chen
145	John	Russell
176	Jonathon	Taylor
112	Jose Manuel	Urman

- *SQL WHERE clause with LIKE operator*

```
SELECT employee_id, first_name, last_name
FROM [dbo].[Employees]
WHERE first_name LIKE '_h%'
```

employee_id	first_name	last_name
179	Charles	Johnson
123	Shanta	Vollman
205	Shelley	Higgins
116	Shelli	Baida



SQL WHERE clause with IS NULL

- To determine whether an expression or column is NULL or not, you use the IS NULL operator.
- To check if an expression or column is not NULL, you use the IS NOT operator:

Example

➢ *SQL WHERE clause with IS NULL operator*

```
SELECT employee_id, first_name, last_name,    phone_number
FROM [dbo].[Employees]
WHERE phone_number IS NULL;
```

employee_id	first_name	last_name	phone_number
145	John	Russell	NULL
146	Karen	Partners	NULL
176	Jonathon	Taylor	NULL
177	Jack	Livingston	NULL
178	Kimberely	Grant	NULL
179	Charles	Johnson	NULL

employees

- * employee_id
- first_name
- last_name
- email
- phone_number
- hire_date
- job_id
- salary
- manager_id
- department_id

➢ *SQL WHERE clause with IS NOT NULL operator*

```
SELECT employee_id, first_name, last_name,    phone_number
FROM [dbo].[Employees]
WHERE phone_number IS NOT NULL;
```

employee_id	first_name	last_name	phone_number
100	Steven	King	515.123.4567
101	Neena	Kochhar	515.123.4568
102	Lex	De Haan	515.123.4569
103	Alexander	Hunold	590.423.4567
104	Bruce	Ernst	590.423.4568
105	David	Austin	590.423.4569
106	Valli	Pataballa	590.423.4560
107	Diana	Lorentz	590.423.5567

SQL ORDER BY clause

- The ORDER BY is an optional clause of the SELECT statement.
- The ORDER BY clause allows you to sort the rows returned by the SELECT clause by one or more sort expressions in ascending or descending order.

Syntax:

```
SELECT column_list
FROM table1
ORDER BY sort_expression [ASC | DESC];
```

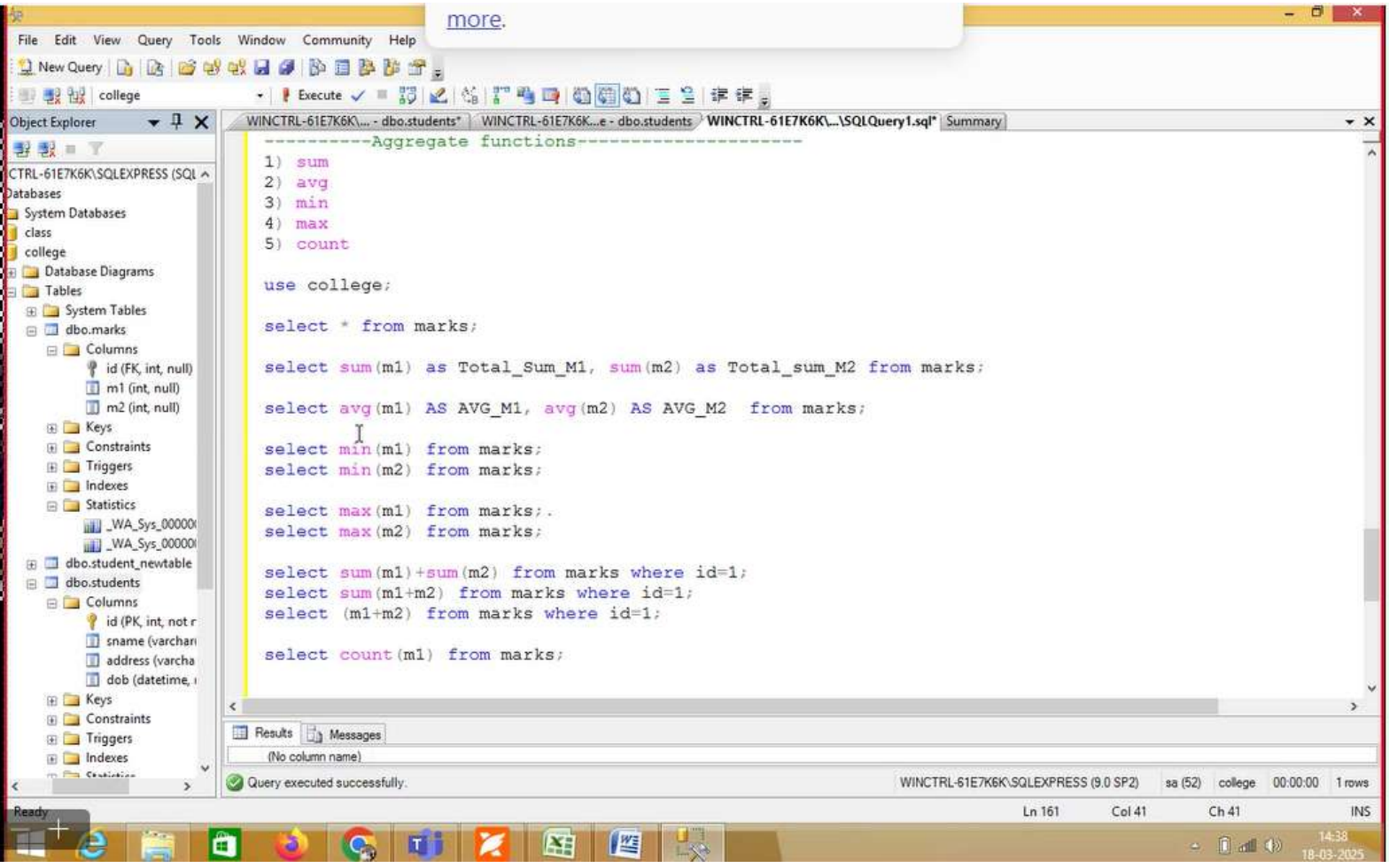
SQL LIMIT and OFFSET clause

- To limit the number of rows returned by a select statement, you use the FETCH and OFFSET clauses.

Syntax:

```
ORDER BY column_list [ASC |DESC]
OFFSET offset_row_count {ROW | ROWS}
FETCH {FIRST | NEXT} fetch_row_count {ROW | ROWS} ONLY;
```

- The FETCH row_count determines the number of rows (row_count) returned by the query.
- The OFFSET offset clause skips the offset rows before beginning to return the rows. The OFFSET clause is optional.


more.
File Edit View Query Tools Window Community Help
New Query
college
Execute
Object Explorer
WINCTRL-61E7K6K\... - dbo.students*
WINCTRL-61E7K6K...e - dbo.students
WINCTRL-61E7K6K\...\SQLQuery1.sql*
Summary
----------Aggregate functions----------
1) sum
2) avg
3) min
4) max
5) count
use college;
select * from marks;
select sum(m1) as Total_Sum_M1, sum(m2) as Total_sum_M2 from marks;
select avg(m1) AS AVG_M1, avg(m2) AS AVG_M2 from marks;
select min(m1) from marks;
select min(m2) from marks;
select max(m1) from marks;.
select max(m2) from marks;
select sum(m1)+sum(m2) from marks where id=1;
select sum(m1+m2) from marks where id=1;
select (m1+m2) from marks where id=1;
select count(m1) from marks;
Results
Messages
(No column name)
Query executed successfully.
WINCTRL-61E7K6K\SQLEXPRESS (9.0 SP2)
sa (52)
college
00:00:00
1 rows
Ready
Ln 161
Col 41
Ch 41
INS
14:38
18-03-2025

Introduction to Subquery

- A subquery is a query nested within another query such as SELECT, INSERT, UPDATE or DELETE. Also, a subquery can be nested within another subquery.
- A subquery is called an inner query while the query that contains the subquery is called an outer query.
- A subquery can be used anywhere that expression is used and must be closed in parentheses.

- You can use a subquery in many places such as:
  - With the IN or NOT IN operator
  - With comparison operators
  - With the EXISTS or NOT EXISTS operator