# **Лекція 11. Елементи мови SQL (продовження 2)**

# **Використання об’єднання, перетину та різниці**

**Приклад30**. Отримати імена постачальників, які мають статус, більше 3 або такі, що постачають хоча б одну деталь номер 2 (об’єднання двох підзапитів - ключове слово ***UNION***):

SELECT P.PNAME FROM P

WHERE P.STATUS > 3 UNION

SELECT P.PNAME

FROM P, PD

WHERE P.PNUM = PD.PNUM AND

PD.DNUM = 2;

Зауважання. Вихідні таблиці об’єднуючих запитів повинні бути сумісними, тобто мати однакову кількість стовбців та однакові типи стовбців у порядку їх перерахування. *Не потрібно*, щоб об’єднувальні таблиці мали однакові імена колонок. Це відрізняє операцію об’єднання запитів в SQL від операції об’єднання в реляційній алгебрі. Найменування колонок в результатуючому запиті будуть автоматично взяті з результату першого запиту в об’єднанні.

**Приклад31**. Отримати імена постачальників, які мають статус, більше 3 та одночасно такі, що постачають хоча б одну деталь номер 2 (перетин двох підзапитів - ключове слово ***INTERSECT***):

SELECT P.PNAME FROM P

WHERE P.STATUS > 3 INTERSECT

SELECT P.PNAME FROM P, PD

WHERE P.PNUM = PD.PNUM AND

PD.DNUM = 2;

**Приклад32**. Отримати імена постачальників, які мають статус, більше 3, за виключенням тих, хто поставляє хоча б одну деталь номер 2 (різниця двох підзапитів - ключове слово ***EXCEPT***):

SELECT P.PNAME FROM P

WHERE P.STATUS > 3

EXCEPT

SELECT P.PNAME

FROM P, PD

WHERE P.PNUM = PD.PNUM AND

PD.DNUM = 2;

# **Синтаксис оператора вибірки даних (SELECT) у BNF - нотаціях**

Задамо синтаксис оператора вибірки даних (оператора SELECT) з використанням умовних позначень у вигляді ***стандартних форм Бэкуса-Наура*** (***BNF***).

В BNF позначеннях застосовуються такі елементи:

* Символ "::=" означає рівність за означенням. Зліва від знаку стоїть поняття, яке визначається, праворуч - визначення поняття.
* Ключові слова записуються прописними літерами. Вони зарезервовані і складають частину оператора.
* Мітки-заповнювачі конкретних значень елементів та змінних записуються курсивом.
* Необов’язкові елементи оператора заключаються у квадратні дужки [].
* Вертикальна риска | вказує на те, що усі попередні елементи списку є необов’язковими і можуть бути замінені на довільний елемент списку після цєї риски.
* Фигурні дужки {} вказують на те, що все всередині дужок є єдиним цілим.
* Три крапки "…" означають, що попередня частина оператора може бути довільну кількість разів повторена.

Багато точок, всередині яких знаходиться кома".,.." вказує на те, що попередня частина оператора, яка складається з декількох елементів, розділених комами, може мати довольну кількість повторень.

* Круглі дужки є елементом оператора.

# **Синтаксис оператора вибірки**

*Оператор вибірки* ::= *Табличний вираз* [**ORDER BY**

{{*Ім*’*я стовбця-результату* [**ASC** | **DESC**]} | {*Додатнє ціле* [**ASC** | **DESC**]}}.,..];

*Табличний вираз*::= *Select-вираз*

[

{**UNION** | **INTERSECT** | **EXCEPT**} [**ALL**]

{*Select-вираз* | **TABLE** *Ім’я таблиці* | *Конструктор значень таблиці*}

]

*Select-вираз* ::= SELECT **[**ALL **|** DISTINCT**]**

{{{*Скалярний вираз* | *Функція агрегування* | *Select-вираз*} [**AS** *Ім’я стовбця*]}.,..}

| {{*Ім’я таблиці*|*Ім’я кореляції*}.\*}

| \*

# **FROM** {

{*Ім’я таблиці* [**AS**] [*Імя кореляції*] [(*Ім’я стовбця*.,..)]}

| {*Select-вираз* [**AS**] *Ім’я кореляції* [(*Ім’я стовбця*.,..)]}

| *Поєднана таблиця* }.,..

[**WHERE** *Умовний вираз*]

[**GROUP BY** {[{*Ім’я таблиці*|*Ім’я кореляції*}.]*Ім’я стовбця*}.,..]

[**HAVING** *Умовний вираз*]

Зауваження. Select-вираз в розділі SELECT повинен повертати таблицю, яка складається з одного рядка та одного стовбця, тобто скалярний вираз.

Зауваження. Умовний вираз в разділі WHERE повинен обраховуватися для кожного рядка, який є кандидатом до результуючої множини рядків. В умовному виразі можна використовувати підзапити. Синтаксис умовних виразів для розділу WHERE наведено далі.

Зауваження. Розділ HAVING містить умовний вираз, який обчислюється для кожної групи, що визначається списком групування в розділі GROUP BY. Це умовний вираз може містити функції агрегування, які обчислюютьс для кожної групи. Умовний вираз, який сформульований у розділі WHERE, може бути перенесений в розділ HAVING..

Зауваження. Якщо в розділі SELECT присутні агрегатні функції, то вони обчислюються по-різному, в залежності від наявності розділу GROUP BY. Якщо роздел GROUP BY відсутній, то результат запиту повертає не більш одного рядка. Агрегатні функції обчислюються за всіма рядками, що задовільняють умовному виразу в розділі WHERE. Якщо розділ GROUP BY присутній, то агрегатні функції обчислюються окремо для кожної групи, що визначена в розділі GROUP BY.

*Скалярний вираз* - в якості скалярних виразів в розділі SELECT можуть розглядатися або імена стовбців таблиць, які входять в розділ FROM, або прості функції, що повертають скалярні значення.

Функція агрегування ::= **COUNT** (\*) | {

{**COUNT** | **MAX** | **MIN** | **SUM** | **AVG**} ([**ALL** | **DISTINCT**] *Скалярное выражение*)

}

*Конструктор значень таблиці* ::=

**VALUES** *Конструктор значень рядка*.,..

*Конструктор значень рядка* ::=

*Елемент конструктора* | (*Елемент конструктора*.,..) | *Select-вираз*

Зауваження. Select-вираз, який використовується в конструкторі значень рядка, повинен повертати рівно один рядок.

*Елемент конструктора* ::=

*Вираз дляобчислення значень* | **NULL** | **DEFAULT**

# **Синтаксис з’єднаних таблиць**

В розділі FROM оператора SELECT можна використовувати з’єднані таблиці. Припустимо, що в результаті деяких операцій отримано таблиці A та B. Тоді синтаксис з’єднаної таблиці має такий вигляд:

*З*’*єднана таблиця* ::=

*Перехресне з*’є*днання*

| *Природне з*’*єднання*

| *З*’*єднання за допомогою предикату*

| *З*’*єднання за допомогою імен стовбців*

| *З*’*єднання об’єднання*

Тип з’єднання ::= **INNER** | **LEFT** [**OUTER**] | **RIGTH** [**OUTER**] | **FULL** [**OUTER**]

*Перехресне з*’*єднання*::= *Таблиця А* **CROSS JOIN** *Таблиця В*

*Природне з*’*єднання* ::=

*Таблица А* [**NATURAL**] [*Тип з*’*єднання*] **JOIN** *Таблица В*

*З*’*єднання за допомогою предикату* ::=

*Таблиця А* [*Тип з’єднання*] **JOIN** *Таблиця В* **ON** Предикат

*З*’*єднання за допомогою імен стовбців* ::=

*Таблиця А* [*Тип з’єднання*] **JOIN** *Таблиця В* **USING** (Ім’я стовбця.,..)

*З*’*єднання об’єднанням* ::=

*Таблиця А* **UNION JOIN** *Таблиця В*

де:

CROSS JOIN - перехресне з’єднання повертає декартовий добуток таблиць. Таке з’єднання в розділі FROM може бути замінено на список таблиць через кому.

NATURAL JOIN – природне з’єднання відбувається за усіма стовбцями таблиць А і В, що мають однакові імена. В результуючу таблицю однакові стовбці вставляються лише один раз.

JOIN … ON - з’єднання за допомогою предикату з’єднує рядки таблиць А і В за допомогою вказаного предикату.

JOIN … USING - з’єднання за допомогою імен стовбців з’єднує відношення подібно до природнього з’єднання за тими спільними стовбцями таблиць А і В, які вказано у списку USING.

OUTER - ключове слово OUTER (зовнішній) не є обов’язковим, воно не застосовується в жодних операціях з даними.

INNER - тип з’єднання "внутрішне". Внутрішній тип з’єднання використовується за замовчанням, коли тип явно не задано. В таблицях А і В з’єднуються лише ті рядки, для яких знайдено співпадіння.

LEFT (OUTER) - тип з’єднання "ліве (зовнішне)". Ліве з’єднання таблиць А і В включає в себе усі рядки з лівої таблиці А та ті рядки з правої таблиці В, для яких знайдено співпадіння. Для рядків з таблиці А, для яких не знайдено відповідності в таблиці В, у стовбці, що виділяються з таблиці В, заносяться значення NULL.

RIGHT (OUTER) - тип з’єднання "праве (зовнішне)". Праве з’єднання таблиць А і В включає в себе усі рядки з правої таблиці В та ті рядки з лівої таблиці А, для яких знайдено співпадіння. Для рядків таблиці В, для яких не знайдено відповідності в таблиці А, у стовбці, що виділяються з таблиці А заносяться значення NULL.

FULL (OUTER) - тип з’єднання "повне (зовнішне)". Це комбінація лівого та правого з’єднань. В повне з’єднання включаються усі рядки з обох таблиць. Для рядків, що співпали, поля заповнюються реальними значеннями, для рядків, що не співпали, поля заповнюються у відповідності з правилами лівого та правого з’єднань.

UNION JOIN - з’єднання об’єднанням є оберненим по відношенню до внутрішнього з’єднання. Воно включає лише ті рядки з таблиць А і В, для яких не знайдено співпадінь. В них використовуються значення NULL для стовбців, які отримано з другої таблиці. Якщо взяти повне зовнішне з’єднання і видалити з нього рядки, які отримано в результаті внутрішнього з’єднання, то получається з’єднання об’єднанням.

Застосування з’єднаних таблиць часто полегшує сприйняття оператора SELECT, особливо, коли використовується природне з’єднання. Якщо не використовувати з’єднані таблиці, то при виборі даних з декількох таблиць необхідно явно вказувати умови з’єднання в розділі WHERE. Якщо при цьому користувач вказує складні критерії відбору рядків, то у розділі WHERE змішуються семантично різні поняття - як умови зв’язку таблиць, так і умови відбору рядків.

# **Синтаксис умовних виразів розділу WHERE**

Умовний вираз, який використовується у розділі WHERE оператора SELECT повинен обчислюватися для кожного рядка-кандидата, що відбирається оператором SELECT. Умовний вираз може повертати одне з трьох значень істинності: TRUE, FALSE або UNKNOWN. Рядок-кандидат відбирається в результуючу множину рядків лише у тому випадку, якщо для нього умовний вираз повернув значення TRUE.

Умовні вирази мають такий синтаксис:

*Умовний вираз* ::= [ ( ] [**NOT**]

{*Предикат порівняння*

| *Предикат between*

| *Предикат in*

| *Предикат like*

| *Предикат null*

| *Предикат кількісного порівняння*

| *Предикат exist*

| *Предикат unique*

| *Предикат match*

| *Предикат overlaps*}

[{**AND** | **OR**} *Умовний вираз*] [ ) ]

[**IS** [**NOT**] {**TRUE** | **FALSE** | **UNKNOWN**}]

*Предикат порівняння* ::=

*Конструктор значень рядка* {= | < | > | <= | >= | <>} *Конструктор значень рядка*

**Приклад 33**. Порівняння поля таблиці та скалярного значення:

POSTAV.VOLUME > 100

**Приклад 34**. Порівняння двох сконструйованих рядків:

(PD.PNUM, PD.DNUM) = (1, 25)

Цей приклад еквівалентний умовному виразу

PD.PNUM = 1 AND PD.DNUM = 25

*Предикат between* ::=

*Конструктор значень рядка* [**NOT**] **BETWEEN**

*Конструктор значень рядка* **AND** *Конструктор значень рядка*

**Приклад 35**. PD.VOLUME BETWEEN 10 AND 100

*Предикат in* ::=

*Конструктор значень рядка* [**NOT**] **IN**

{(*Select-вираз*) | (*Вираз для обчислення значення*.,..)}

# **Приклад 36**.

P.PNUM IN (SELECT PD.PNUM FROM PD WHERE PD.DNUM=2)

# **Приклад 37**.

* 1. NUM IN (1, 2, 3, 5)

*Предикат like* ::=

*Вираз для обчислення значення рядка-пошуку* [**NOT**] **LIKE**

*Вираз для обчислення значення рядка-шаблона* [**ESCAPE** *Символ*]

Зауваження. Предикат LIKE здійснює пошук рядка-пошуку у рядку-шаблоні. В рядку- шаблоні дозволяється використовувати два трафаретних символи:

* + - Символ підкреслення "\_" може використовуватися замість довільного одиничного символу в рядку-пошуку,
    - Символ відсотку "%" може заміняти набір будь-яких символів у рядку-пошуку

(кількість символів в наборі може бути від 0 і більше).

*Предикат null* ::=

*Конструктор значень рядка* **IS** [**NOT**] **NULL**

Зауваження. Предикат NULL застосовується спеціально для перевірки на null значення виразу, який перевіряється.

*Предикат кількісного порівняння* ::=

*Конструктор значень рядка* {= | < | > | <= | >= | <>}

{**ANY** | **SOME** | **ALL**} (*Select-вираз*)

Зауваження. Квантори ANY та SOME є синонімами та повністю замінюють один інший.

Зауваження. Якщо вказано один з кванторів ANY і SOME, то предикат кількісного порівняння повертає TRUE, якщо значення, що порівнюється, співпадає *хоча б з одним* значенням, яке повертається в підзапиті (select-виразі).

Зауваження. Якщо вказано квантор ALL, то предикат кількісного порівняння повертає TRUE, якщо значення, що порівнюється, співпадає  *з кожним* значенням, яке повертається в підзапиті (select-виразі).

# **Приклад 38**.

P.PNUM = SOME (SELECT PD.PNUM FROM PD WHERE PD.DNUM=2)

*Предикат exist* ::=

**EXIST** (*Select-вираз*)

Зауваження. Предикат EXIST повертає значення TRUE, якщо результат підзапиту

(select-виразу) не порожній.

*Предикат unique* ::=

**UNIQUE** (*Select-вираз*)

Зауваження. Предикат UNIQUE повертає TRUE, якщо у результаті підзапиту (select-

виразу) немає рядків, які співпадають.

*Предикат match* ::=

*Конструктор значень рядка* **MATCH** [**UNIQUE**]

[**PARTIAL** | **FULL**] (*Select-вираз*)

Зауваження. Предикат MATCH перевіряє, чи буде значення, яке визначено в конструкторі рядка співпадати зі значенням будь-якого рядка, отриманого в результаті підзапиту.

*Предикат overlaps* ::=

*Конструктор значень рядка* **OVERLAPS** *Конструктор значень рядка*

Зауваження. Предикат OVERLAPS є спеціалізованим предикатом, що дозволяє визначити, чи буде вказаний період часу перекривати інший період часу.

# **Порядок виконання оператора SELECT**

Розглянемо концептуальну схему виконання оператора SELECT. Ця схема є концептуальною, тому що гарантується, що результат буде таким, якщо б він виконувався крок за кроком у відповідності з цією схемою. Насправді, реальний результат розраховується за допомогою більш складних алгоритмів, якими "володіє" конкретна СУБД.

# **Стадія 1. Виконання одиночного оператору SELECT**

Якщо в операторі присутні ключові слова UNION, EXCEPT та INTERSECT, то запит розбивається на декілька незалежних запитів, кожний з яких виконується окремо:

*Крок 1 (FROM)*. Обчислюємо прямий декартовий добуток усіх таблиць, вказаних у обов’язковому розділі FROM. В результаті кроку 1 отримуємо таблицю A.

*Крок 2 (WHERE)*. Якщо в операторі SELECT присутній розділ WHERE, то сканується таблиця A, отримана на кроці 1. При цьому для кожного рядка з таблиці A обчислюється умовний вираз, наведений в розділі WHERE. Лише ті рядки, для яких умовний вираз повертає значення TRUE, попадає у результат. Якщо розділ WHERE відсутній, то одразу переходимо на крок 3. Якщо в умовному виразі присутні вкладені підзапити, то вони обчислюються у відповідності з даною концептуальною схемою. В результаті кроку 2 отримуємо таблицю B.

*Крок 3 (GROUP BY)*. Якщо в операторі SELECT присутній розділ GROUP BY, то рядки таблиці B, отриманої на другому кроці, групуються у відповідності зі списком групування, який наведено в розділі GROUP BY. Якщо розділ GROUP BY відсутній, то одразу переходимо на крок 4. В результаті кроку 3 отримуємо таблицю С.

*Крок 4 (HAVING)*. Якщо в операторі SELECT присутній розділ HAVING, то групи, які не задовольняють умовний вираз, наведений у розділі HAVING, відкидаються. Якщо розділ HAVING відсутній, то одразу переходимо на крок 5. В результаті кроку 4 отримуємо таблицю D.

*Крок 5 (SELECT)*. Кожна група, яку отримано на кроці 4, генерує один рядок результату таким чином. Обчислюються усі скалярні вирази, що вказані в розділі SELECT. За правилами застосування розділу GROUP BY, такі скалярні вирази повинні бути однаковими для усіх рядків всередині кожної групи. Для кожної групи обчислюються значення агрегатних функцій, наведених у розділі SELECT. Якщо розділ GROUP BY відсутній, але в розділі SELECT є агрегатні функції, то вважається, що є лише одна група. Якщо немає ні розділу GROUP BY, ні агрегатних функцій, то вважається, що є стільки груп, скільки рядків відібрано на даний момент. В результаті кроку 5 отримуємо таблицю E, яка містить стільки колонок, скільки елементів наведено у розділі SELECT та стільки рядків, скільки відібрано груп.

# **Стадія 2. Виконання операцій UNION, EXCEPT, INTERSECT**

Якщо в операторі SELECT присутні ключові слова UNION, EXCEPT та INTERSECT, то над таблицями, отриманими в результаті виконання 1-й стадії, відбувається побудова об’єднання, віднімання або перетину.

# **Стадія 3. Впорядкування результату**

Якщо у операторі SELECT присутній розділ ORDER BY, то рядки отриманої на попередніх кроках таблиці впорядковуються у відповідності зі списком впорядкування, наведеному в розділі ORDER BY.

# **Яким чином насправді виконується оператор SELECT**

Якщо уважно розглянути наведений вище концептуальний алгоритм обчислення результату оператора SELECT, то одразу зрозуміло, що виконувати його безпосередньо в такому вигляді надзвичайно важко. Навіть на першому кроці, коли обчислюється декартовий добуток таблиць, наведений у розділі FROM, може бути отримана таблиця дуже великого розміру, причому практично більшість рядків і колонок з неї буде відкинуто далі на наступних кроках.

Насправді у РСУБД є ***оптимізатор***, функцією якого є знаходження такого *оптимального алгоритму* виконання запиту, який гарантує отримання правильного результату.

Схематично роботу оптимізатора можна подати у вигляді послідовності декількох кроків:

*Крок 1 (Синтаксичний аналіз)*. Запит, яки поступає, підлягає синтаксичному аналізу. На цьому кроці визначається, чи правильно взагалі (з точки зору синтаксису SQL) сформульовано запит. У ході синтаксичного аналізу формулюється деяке внутрішнє подання запиту, яке використовується на наступних кроках.

*Крок 2 (Перетворення в канонічну форму)*. Запит у внутрішньому поданні перетворюється у деяку канонічну форму. При перетворенні до канонічної форми використовуються як синтаксичні, так і семантичні перетворення. Синтаксичні перетворення (наприклад, наведення логічних виразів до кон’юнктивної або диз’юнктивної нормальної форми, заміна виразів "x AND NOT x" на "FALSE", і т.і.) дозволяють отримати нове внутрішнє представлення запиту, яке синтаксично *еквівалентне* початковому, але стандартне в деякому розумінні. Семантичні перетворення використовують додаткові знання, якими володіє система, наприклад, обмежень цілісності. В результаті семантичних перетворень отримуємо запит, який синтаксично *не є еквівалентним* початковому, але такий, що дає *той самий результат*.

*Крок 3 (Генерація планів виконання запиту і вибір оптимального плану)*. На цьому кроці оптимізатор генерує множину можливих планів виконання запиту. Кожен план будується як комбінація низькорівневих процедур доступу до даних з таблиць, методам з’єднання таблиць. З усіх сгенерованих планів обирається план мінімальної вартості. При цьому аналізуються дані про наявність індексів у таблиці, статистичні дані про розподіл значень у таблицях, і т.і. Вартість плану - це, як правило, сума вартостей виконання окремих низькорівневих процедур, які використовуються для його виконання. У вартість виконання окремої процедури можуть входити оцінки кількості звертань до жорсткого диску, степінь завантаженості процесора та інші параметри.

*Крок 4. (Виконання плану запиту)*. На цьому кроці план, обраний на попередньому кроці, передається на реальне виконання.

Якість конкретної СУБД визначається якістю її оптимізатора. Хороший оптимізатор може підвищити швидкість виконання запиту на декілька порядків. Якість оптимізатора визначається тим, які методи перетворень він може використовувати, якою статистичною та іншою інформацією про таблиці він володіє, які методи для оцінки вартості виконання плану він знає.