МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ЧЕРНІГІВСЬКИЙ НАЦІОНАЛЬНИЙ ТЕХНОЛОГІЧНИЙ УНІВЕРСИТЕТ
НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ ТЕХНОЛОГІЙ

Бази даних

Методичні вказівки
до виконання лабораторних робіт
для студентів напряму підготовки
6.050103 – "**Програмна інженерія**"
Частина 1

Обговорено і рекомендовано
на засіданні кафедри
інформаційних технологій та
програмної інженерії
протокол № 21 від 13.06.2016

Чернігів ЧНТУ 2016

Бази даних. Методичні вказівки до виконання лабораторних робіт для студентів напряму підготовки 6.050103 – "Програмна інженерія". Частина 1. /Укл.: Бальченко І.В. – ЧНТУ, 2016. – 57с.

Укладач: Бальченко Ірина Володимирівна, кандидат технічних наук, асистент кафедри інформаційних технологій та програмної інженерії

Відповідальний за випуск: Литвинов В.В., доктор технічних наук, професор, завідувач кафедри інформаційних технологій та програмної інженерії

Рецензент: Зайцев С.В., кандидат технічних наук, доцент, завідувач кафедри інформаційних та комп'ютерних систем Чернігівського національного технологічного університету

Зміст

ВСТУП

Лабораторні роботи є сполучною ланкою між лекційними заняттями і самостійною роботою студентів. В процесі виконання лабораторних робіт експериментально перевіряються ключові питання курсу «Бази даних», набуваються практичні навички проектування і розробки баз даних засобами сучасних систем управління базами даних, перевіряється рівень засвоєння основних положень предмету.

Лабораторні роботи виконуються на персональних комп'ютерах з використанням системи управління баз даних PostgreSQL. Передбачається, що студенти знайомі з основами роботи на персональному комп'ютері. Необхідно володіти мишкою і клавіатурою, вміти маніпулювати файлами, знати який-небудь текстовий редактор. Якщо таких навичок немає, то студент повинен їх набути під час самостійної роботи в лабораторії. Можна скористатися методичними вказівками «Основи роботи на персональному комп'ютері» для студентів економічних спеціальностей. Передбачається, що студенти володіють англійською мовою в обсязі програми середньої школи.

Студент зобов'язаний до лабораторного заняття прочитати методичні вказівки до лабораторної роботи і спробувати виконати її самостійно. Під час лабораторного заняття студент показує викладачеві результати роботи, проводить консультації з виниклих питань та завершує роботу. Обсяг виконаної роботи може бути різним, залежно від того, на яку оцінку претендує студент. Коли робота закінчена, студент повинен її захистити. Захист полягає в виконанні практичного завдання, відповіді на питання по темі лабораторної роботи і внесення деяких змін в розроблюваній базі даних в присутності викладача.

По кожній роботі студент повинен оформити звіт. Звіти оформляються за допомогою текстового редактора Word на папері формату А4 відповідно до вимог стандартів на оформлення технічної документації. Звіт по роботі є розділом підсумкового документа. В кінці семестру звіти підшиваються в єдиний підсумковий документ в титульному листі, який підписується у керівника, після чого студент отримує допуск до іспиту.

За лабораторну роботу студент може отримати до 100 балів, з урахуванням своєчасності і якості виконання всіх складових роботи. Складовими є: звіт, проект, практичне завдання і відповіді на контрольні питання. Оцінки, отримані за лабораторні роботи, враховуються при виставленні підсумкової оцінки з модулю і на іспиті. Для отримання допуску на іспит всі лабораторні роботи повинні бути виконані і кожна з них оцінена не менше ніж на 60 балів.

Лабораторна робота №1

ІНФОЛОГІЧНА МОДЕЛЬ ПРЕДМЕТНОЇ ОБЛАСТІ

Опис предметної області, виділення об'єктів, атрибутів і обмежень цілісності, визначення зв'язків між об'єктами.

1.1 Теоретичні відомості

База даних – сукупність структурованих, взаємопов'язаних, динамічно оновлюваних даних предметної області. Інфологічна модель предметної області може бути показана як сукупніть об'єктів предметної області та зв'язків між об'єктами.

Кожен об'єкт можна охарактеризувати атрибутами, на підставі яких можна описати ймовірні екземпляри об'єктів.

Виділяють наступні види взаємозв'язків між об'єктами:

- «один до одного» (1:1);
- «один до багатьох» (1:M, M:1);
- «багато до багатьох» (M:N).

1.2 Приклад створення інфологічної моделі

Необхідно розробити базу даних книгарні (назвемо її BookShop), що зберігає дані про книги, постачальників і замовників (покупців).

Крок 1: Виокремлення об'єктів предметної області

Визначимо об’єкти предметної області:

- Книга;
- Постачальник;
- Замовник.

Визначимо початкові зв’язки між об’єктами (рис. 1.1).

Рисунок 1.1 – Модель предметної області «BookShop»

Як видно, об'єкти «книга» і «замовник», а також об'єкти «книга» і «постачальник» знаходяться в зв'язку «багато до багатьох». Зв'язок «багато до багатьох» не реалізується в реляційних моделях систем управління базами даних, які широко представлені на сучасному ринку. Тому ці зв'язки необхідно перетворити до відносин «один до багатьох» шляхом введення додаткового об'єкта в кожному такому зв'язку:

- Факт замовлення;
- Факт поставки;

Тоді остаточна модель предметної області буде виглядати, як показано на рис. 1.2.

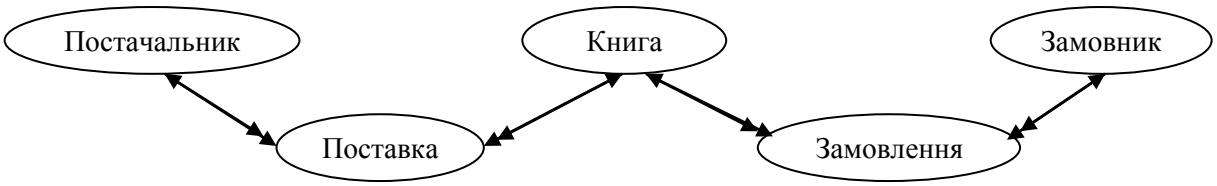

Рисунок 1.2 – Уточнена модель предметної області «BookShop»

Крок 2: Опис атрибутів та обмежень цілісності

Описуємо об’єкти з атрибутами та обмеженнями цілісності (табл. 1.1).

Таблиця 1.1 – Опис атрибутів об’єктів «Bookshop»

Об’єкт	**Атрибут**	**Тип**	**Обмеження**
Книга	Код_книги	Число	унікальне, ключове поле
	Автор	Строка	розміром до 80 символів
	Назва	Строка	розміром до 160 символів
	Видавництво	Строка	розміром до 80 символів
	Ціна	Число в грошовому запису	
	Залишок	Число	[0..1000]
Постачальник	Код_постачальника	Число	унікальне, ключове поле
	Назва	Строка	розміром до 40 символів
	Місто	Строка	розміром до 40 символів
	Адреса	Строка	розміром до 80 символів
	Телефон	Строка	розміром 13 символів
Замовлення	Код_замовлення	Число	унікальне, ключове поле
	Код_книги	Число	зовнішнє посилання
	Код_замовника	Число	зовнішнє посилання
	Сплачено	Строка	розміром до 3 символів, приймає значення «так», «ні»
	Дата_замовлення	Дата	старше ”01/01/2016” за замовчуванням поточна дата
Замовник	Код_замовника	Число	унікальне, ключове поле
	Ім’я	Строка	розміром до 40 символів
	Адреса	Строка	розміром до 80 символів
	Телефон	Строка	розміром 13 символів
Поставка	Номер	Число	унікальне, ключове поле
	Код_книги	Число	зовнішнє посилання
	Код_постачальника	Число	зовнішнє посилання
	Кількість	Число	[0..10000]
	Дата_поставки	Дата	старше ”01/01/2016” за замовчуванням поточна дата

Крок 3: Опис екземплярів об’єктів

Наведемо приклад кількох довільних рядків в об'єктах бази даних BookShop з урахуванням певних обмежень на значення стовпців.

Екземпляри об’єкту «Книга»:

Код книги	**Автор**	**Назва**	**Видавництво**	**Ціна**	**Залишок**
1	К.Дж. Дейт	Введення в системи баз даних	Вільямс	240,00	39
2	Дж. Колби	SQL для починаючих	Вільямс	150,00	48
3	Роберт Дж. Мюллер	Бази даних та UML. Проектування	Лорі	130,00	7

Екземпляри об’єкту «Постачальник»:

Код постачаль-ника	Назва	Місто	Адреса	Телефон
1	ЗАТ «Юнітрейд»	Дніпро	вул. Слобідська 7	097 111-11-11
2	ПАТ «Інстаграм»	Черкаси	пр. Волі 234	067 111-11-11
3	ЗАТ «Поліграф»	Чернігів	пр. Миру 135	050 111-11-11

Екземпляри об’єкту «Замовник»:

Код замовника	Ім’я	Адреса	Телефон
1	Петров І.І.	м. Чернівці, вул. Сілого 7 кв. 54	050 121-11-11
2	Сидоров П.П.	м. Ужгород, вул. Полуботка 17 кв. 154	067 131-11-11
3	Іванов С.С.	м. Ніжин, пр. Перемоги 71 кв. 154	093 151-11-11

Екземпляри об’єкту «Замовлення»:

Код замовлення	Код_книги	Код_замовника	Сплачено	Дата
1	1	2	так	02/10/2014
2	2	3	так	03/10/2014
3	2	2	ні	06/10/2014
4	3	1	ні	08/10/2014

Екземпляри об’єкту «Поставка»:

Номер	Код_книги	Код_постачальника	Кількість	Дата
1	1	1	17	01/10/2014
2	1	2	23	02/10/2014
3	2	3	50	05/10/2014
4	3	1	8	07/10/2014

1.3 Контрольне завдання

1.3.1 Використовуючи метод виокремлення об’єктів предметної області, розробити інфологічну модель бази даних, визначити атрибути та обмеження цілісності згідно варіантів завдань, представлених у таблиці 1.2.

1.3.2 Представити опис атрибутів розробленої інфологічної моделі (з описом обмежень атрибутів).

1.3.3 Привести опис екземплярів об’єктів розробленої моделі (не менше 3 екземплярів кожного об’єкту).

Таблиця 1.2 – Опис варіантів завдань предметної області

№	**Назва розроблюваної БД** **Атрибути універсального відношення**
1	Торгівля
	Код виробу, найменування, марка, виробник, номер прибуткової накладної, дата надходження, на склад, кількість одиниць, закупівельна ціна, роздрібна ціна, номер рахунку-фактури, дата продажу, кількість проданих одиниць, сума, тип платежу готівковий / безготівковий, банківські реквізити покупця для безготівкового розрахунку.
2	Комунальні платежі
	Код послуги, найменування, одиниця виміру, тарифна зона, вартість одиниці, особовий рахунок клієнта, ПІБ клієнта, адреса, телефон, місяць / рік, обсяг споживання послуги, сума до оплати, сума заборгованості.
3	Послуги
	Особистий номер клієнта, ПІБ клієнта, дата народження, домашня адреса, телефон, дата і час прийому замовлення, тип роботи, що виконується, тариф, особистий номер майстра, ФІО майстра, номер ордера на виконання замовлення, дата і час початку роботи, дата і час закриття ордера, обсяг виконаної роботи, вартість.
4	Нарахування зарплатні
	Номер особової справи, ПІБ співробітника, домашня адреса, домашній телефон, робочий телефон, дата прийому, на роботу, стаж роботи за фахом, загальний стаж роботи, освіта, кваліфікація, посада, ставка заробітної плати, місяць / рік, кількість робочих днів за місяць , фактично відпрацьованих днів, преміальні, відпускні, утримання в процентах від нарахування, аванс, сума до видачі.
5	Поставки
	Код продукції, найменування продукції, одиниця виміру, код виробника, місяць і рік випуску, код постачальника, юридичне найменування постачальника, юридична адреса постачальника, банківські реквізити постачальника, телефон постачальника, номер договору на поставку, дата підписання договору, термін поставки, кількість одиниць, показник якості, оптова ціна, умови поставки.
6	Білетна каса
	Номер рейсу, пункт відправлення, пункт призначення, час відправлення, час прибуття, категорія квитка, вартість квитка, тип транспортного засобу, загальна кількість місць, кількість місць даної категорії, унікальний порядковий номер пасажира, ФІО пасажира, дата і час відправлення по квитку, дата і час продажу квитка, дата і час бронювання квитка.
7	Відділ кадрів
	Особистий номер співробітника, ПІБ співробітника, номер паспорта, дата народження, домашня адреса, домашній телефон, освіта, номер диплома, код спеціальності, найменування спеціальності, кваліфікація, код посади, найменування посади, посадовий оклад, дата проходження підвищення кваліфікації, присвоєно кваліфікацію, свідоцтво про підвищення кваліфікації, дата заохочення, вид заохочення, дата накладеного стягнення, вид стягнення.
8	Готель
	Номер паспорта клієнта, ПІБ, дата народження, громадянство, номер кімнати, категорія, вартість проживання за добу, дата і час поселення, дата і час виселення, дата бронювання, сума до оплати, особистий номер адміністратора, ПІБ адміністратора.

Продовження таблиці 1.2

№	**Назва розроблюваної БД** **Атрибути універсального відношення**
9	Виробництво
	Код продукції, найменування, код сировини, найменування сировини, норма витрат сировини для виробництва одиниці продукції, відсоток втрат при виготовленні продукції, собівартість продукції, оптова ціна, рік випуску, план випуску, фактично вироблено, відсоток бракованої продукції, валовий дохід, чистий прибуток, номер експортної партії, дата відправлення на експорт, найменування імпортера, країна імпортера, кількість продукції в партії.
10	Банк
	Код клієнта, юридичне найменування клієнта, юридична адреса клієнта, ПІБ директора, телефон директора, ПІБ головного бухгалтера, телефон бухгалтера, номер рахунку клієнта, залишок на рахунку, номер рахунку дебітора, банк дебітора, юридичне найменування дебітора, номер рахунку кредитора, банк кредитора, юридичне найменування кредитора, номер договору про кредитування, дата видачі кредиту, термін користування кредитом, сума кредиту, процентна ставка.
11	Експорт/Імпорт
	Код підприємства, назва підприємства, юридична адреса, телефон / факс, WWW, балансовий рік, валовий дохід, чистий прибуток, код товарної номенклатури зовнішньоекономічної діяльності (ТНЗД), найменування ТНЗД, код групи виробництва (ГВ), найменування ГВ, рік операції експорту / імпорту, обсяг експорту / імпорту, рейтинг підприємства.
12	Проект
	Код роботи, найменування роботи, дата початку, найраніша дата початку, дата закінчення, найпізніша дата закінчення, загальний обсяг в людино-годинах, загальна потреба в матеріалах в умовних одиницях, номер віхи, показник обсягу виконаної роботи в процентах до загального обсягу, показник використаних матеріалів у відсотках до загальної потреби, код виконавця, найменування виконавця, юридична адреса виконавця, номер тижня з початку року, обсяг виконаної роботи, кількість використаних матеріалів.

1.3.3 Оформити звіт по результатам виконання контрольного завдання

Звіт повинен містити:

1) інфологічну модель предметної області, що не містить зв’язків типу «багато до багатьох».

2) опис атрибутів об’єктів інфологічної моделі та обмеження цілісності.

1.4 Контрольні запитання

1) Що таке об’єкт предметної області?
2) Як визначити атрибути об’єкта предметної області?
3) Що таке первинний ключ?
4) Що таке складений первинний ключ?
5) Для чого та як визначаються обмеження на значення стовпців?
6) Що таке обмеження первинного ключа?
7) Що таке обмеження зовнішнього ключа?
8) Що таке зв’язування?
9) Що таке логічна модель бази даних?

Лабораторна робота №2

АРХІТЕКТУРА СИСТЕМ УПРАВЛІННЯ РЕЛЯЦІЙНИМИ БАЗАМИ ДАНИХ

Архітектура СУБД, які підтримують реляційну модель даних. Засоби СУБД для створення бази даних і роботи з її об'єктами.

2.1 Теоретичні відомості

2.1.1 Архітектура СУБД

2.1.1.1 Архітектура СУБД MS SQL Server

MS SQL Server – повнофункціональна СУБД масштабу підприємства. Ця система призначена для управління базами даних і їх аналізу і дозволяє швидко розробляти корпоративні бізнес-додатки, орієнтовані на OLAP-аналіз даних на основі використання багатовимірних сховищ даних, електронну комерцію і використання Internet для доступу до даних. Поточна комерційна версія – MS SQL Server 2016. SQL Server 2016 офіційно представлений в чотирьох пакетних рішеннях: Enterprise, Standart, Express і Developer, останні 2 з яких доступні безоплатно. Особливістю даної СУБД є те, що вона Windows-орієнтована.

Фізична архітектура СУБД MS SQL Server

Дані MS SQL Server зберігаються в базах даних, які організовані в логічні компоненти, видимі користувачеві. SQL Server має кілька баз даних. Перш за все, це – чотири системних бази даних: master, model, tempdb і msdb. Крім того, може бути визначено декілька орієнтованих на користувача баз даних.

Незалежно від того, є база даних системною або призначеною для користувача, всі вони складаються з певного набору файлів, які є індивідуальними для кожної бази даних. Кожна база даних має два типи файлів:

– **файли даних** (data files), в яких зберігаються об'єкти бази даних (таблиці, індекси, обмеження та ін.);

– **журнали транзакцій** (transaction log files), куди MS SQL Server записує всі транзакції, що виконуються, перед тим, як записати їх в базу даних.

Логічна архітектура СУБД MS SQL Server

Основними компонентами MS SQL Server є:

– **Databases** – набір таблиць з даними та інші об’єкти, такі як представлення (views), індекси (indexes), збережені процедури (stored procedure), тригеры (trigger) і так далі, что забезпечує додаткові можливості по управлінню даними;

– **Data Transformation Services** – сервіси трансформації даних – засоби, за допомогою яких можливо не тільки трансформувати об'єкти баз даних (індекси, збережені процедури і все таке) з одного комп'ютера на інший, а й проводити імпорт / експорт різнорідних даних, використовуючи архітектуру OLE DB;

– **SQL Server Agent** – багатосерверних майстер, який забезпечує диспетчеризацію завдань (jobs), повідомлень (alerts), операторів (operators), повідомлень (notifications) і реплікації (replication);

– **Backup** – резервування і подальше відновлення баз даних і журналів транзакцій на спеціально призначений для цього логічний пристрій (backup device);

– **Database Maintenance Plans** – майстер планування експлуатації баз даних, за допомогою якого вирішуються завдання реорганізації сторінок даних і індексів, стиснення файлів бази даних, після видалення з неї порожніх сторінок, контролю узгодженості даних, резервування бази даних і журналу транзакцій;

– **SQL Server Logs** – файли (журнали транзакцій), куди SQL Server записує всі транзакції перед тим, як провести зміни в самій базі даних;

– **Web Publishing** – майстер формування файлів HTML з даних SQL Server;

– **Logins** – набір параметрів, що дозволяють визначити ім'я і пароль користувача, а також пов'язані з ними рівні доступу до компонентів бази даних;

– **Server Roles** – засіб, який дозволяє об'єднувати користувачів в групи і призначати їм певні повноваження, причому користувачі можуть «грати» (належати до) різні ролі;

– **Linked Servers** – засіб конфігурації пов'язаних серверів, які зазвичай використовуються для виконання розподілених запитів;

– **Remote Servers** – засіб налаштування віддалених серверів, яке забезпечує можливість користувачам, підключеним до одного сервера, виконувати збережені процедури на інших серверах.

Дані в SQL Server організовані в декількох різних об'єктах, до яких користувач має доступ після підключення до БД. До таких об'єктів належать:

– **Таблиці (tables).** Це єдиний об'єкт бази даних, призначений для зберігання призначених для користувача даних.

– **Представлення (views).** Є *віртуальними таблицями* (virtual tables), які відображають дані, що зберігаються в інших таблицях. Для користувача ж представлення багато в чому нагадують таблиці. Представления використовуються як засіб реалізації зовнішніх моделей користувачів.

– **Індекси (indexes).** Об'єкти цього типу призначені для підвищення продуктивності роботи сервера при пошуку потрібних даних в таблицях і представленнях, що досягається шляхом зберігання в упорядкованому стані даних одного або більше стовпців таблиці або представлення. Таким чином, індекси не можуть існувати самі по собі.

– **Ключі (keys).** Є одним з типів обмеження цілісності. Однак вони грають досить важливу роль в базі даних і тому розглядаються як окремі об'єкти. Проте, реалізуються вони так само, як і інші обмеження цілісності, які зв'язуються з таблицями.

– **Замовчування (defaults).** Цей тип об'єктів описує значення, які привласнюються стовпчикам таблиці, якщо при додаванні рядка явно не було вказано значення для відповідного стовпця.

– **Правила (rules).** Є логічною умовою, що обмежує діапазон можливих значень для стовпця таблиці або визначеного користувачем типу даних.

– **Обмеження цілістності (constraints).** Спеціальні управляючі конструкції, що обмежують діапазон можливих значень в стовпці таблиці. Таким чином, обмеження цілісності виявляються нерозривними з таблицями.

– **Збережені процедури (stored procedures).** Являють собою набір команд SQL, збережених на сервері, і можуть виконуватися як одне ціле. Кожна процедура має своє ім'я. З цього імені користувачі можуть викликати процедуру, запускаючи тим самим на виконання весь набір команд, що представляють тіло процедури. Використання збережених процедур дозволяє знизити вартість супроводу системи і дає можливість позбутися від необхідності змінювати клієнтські програми. Використання збережених процедур також дозволяє значно підвищити безпеку даних. Додаток або користувач отримує лише спеціальне право на виконання збереженої процедури, яка і буде звертатися до даних. Доступу ж до самих даних користувач не отримує.

– **Тригери (triggers).** Це спеціальний тип збережених процедур, що автоматично запускаються сервером при виконанні видалення, вставки або зміни даних в конкретній таблиці. Тобто тригери зв'язуються з певною таблицею і не можуть існувати самі по собі.

– **Визначені користувачем типи даних (user-defined data types, UDDT).** Ці об'єкти мають окреслити основні дані, створювані користувачами.

– **Визначені користувачем функцшї (user-defined function).** Об'єкти цього типу являють собою набір команд Transact-SQL, збережених користувачем у вигляді функції.

– **Ролі бази даних (database roles).** Власне ролі бази даних (фіксовані і призначені для користувача), призначені для угруповання користувачів і надання їм необхідних прав доступу. Це допомагає спростити адміністрування системи безпеки бази даних. Окремо стоять ролі

додатку. Вони потрібні не для угруповання користувачів, а для надання прав доступу.

При створенні нової бази даних SQL Server як відправний пункт використовує шаблон (template) або модель (model) бази даних, яку можна представити у вигляді сукупності встановлених за замовчуванням об'єктів, реалізованих в системних базах даних. Модель бази даних складається зі стандартних об'єктів SQL Server.

Якщо бажано встановити для всіх новостворюваних баз даних певні атрибути, необхідно реалізувати їх в моделі бази даних. Потім при створенні нових баз даних атрибути успадковуються кожною новою базою даних.

Модель бази даних управляється так само, як звичайна база даних, і обмежень на розміщення об'єктів в цій базі даних немає. Для роботи з об'єктами цієї бази даних, як і будь-якою іншою, можна використовувати звичайні способи доступу до БД.

2.1.1.2 Архітектура СУБД PostgreSQL

СУБД PostgreSQL – це кросплатформна вільно поширювана об'єктно-реляційна система управління базами даних, найбільш розвинена з відкритих (open source) СУБД в світі і є реальною альтернативою комерційних баз даних. Поточна версія – PostgreSQL 9.5.

Основні можливості PostgreSQL.

- **Підтримка об'єктно-реляційної моделі**. Робота з даними в PostgreSQL заснована на об'єктно-реляційній моделі, що дозволяє задіяти складні процедури і системи правил. Прикладами нетривіальних можливостей цієї категорії є декларативні запити SQL, контроль паралельного доступу, підтримка багатокористувацького доступу, транзакції, оптимізація запитів, підтримка успадкування і масивів.
- **Простота розширення**. В PostgreSQL підтримуються призначені для користувача оператори, функції, методи доступу і типи даних.
- **Повноцінна підтримка SQL.** PostgreSQL відповідає базовій специфікації SQL 99.
- **Гручкість API**. Гнучкість API PostgreSQL дозволяє легко створювати інтерфейси до СУБД PostgreSQL. В даний час існують програмні інтерфейси для Object Pascal, Python, Perl, PHP, ODBC, Java /JDBC, Ruby, TCL, C/C++ і Pike.
- **Процедурні мови**. У PostgreSQL передбачена підтримка внутрішніх процедурних мов, в тому числі спеціалізованого мови PL/pgSQL, що є аналогом PL/SQL, процедурного мови Oracle. Однією з переваг PostgreSQL є можливість використання Perl, Python і TCL як внутрішніх процедурних мов.
- **Технологія MVCC**. MVCC (Multiversion Concurrency Control) використовується в PostgreSQL для управління конкурентним доступом до даних на різноманітної основі. Ця технологія дозволяє запобігати зайвим блокуванням (locking) операцій читання операціями, що виробляють оновлення записів. PostgreSQL відстежує всі транзакції, що виконуються користувачами бази даних, що дозволяє працювати з записами без очікування їх звільнення. На практиці це означає, що при запиті до БД кожна транзакція бачить як би знімок даних (версію) на момент цього знімка, а не поточний стан даних. Таким чином, транзакції захищаються від перегляду незафіксованих даних, які в даний момент можуть тільки формуватися конкурентними транзакціями в тих же самих рядках таблиці. Основна перевага MMVC полягає в тому, що читання даних ніколи не блокує запис, а запис ніколи не блокує читання.

MMVC дозволяє уникати явного блокування на рівні таблиць і окремих записів, яке використовується в традиційних СУБД, і, таким чином, мінімізує блокування даних і збільшує продуктивність в багатокористувацьких системах БД.

Також реалізовано відстеження взаємних блокувань (deadlocks).

- **Клієнт-серверна архітектура**. У PostgreSQL використовується архітектура «клієнт-сервер» з розподілом процесів між користувачами. В цілому вона нагадує методику роботи з процесами в сервері Apache. Головний (master) процес створює додаткові підключення для кожного клієнта, намагається встановити з'єднання з PostgreSQL.
- **Зберігаюча регістрація WAL.** WAL (Write-Ahead Logging) є стандартним методом

для забезпечення цілісності даних. Зберігаюча реєстрація – метод реєстрації (журналювання) транзакцій, при якому запис в журналі робиться до запису даних. Використовується також в MS SQL Server. Суть WAL полягає в тому, що зміни в файлах даних (таблиці і індекси) повинні бути внесені тільки після запису в журнал (log), в якому фіксуються ці зміни. Ця процедура дозволяє не переписувати сторінки даних на диску при кожній транзакції, так як в разі аварії ми зможемо відновити базу даних за допомогою журналу. Механізм WAL забезпечує наступні переваги:

– Підвищення продуктивності роботи СУБД за рахунок того, що записуються тільки внесені зміни без переписування всіх даних в таблицях.

– Підвищення надійності зберігання даних за рахунок попереднього збереження буферизованих даних в WAL.

– Можливість відкату стану БД на будь-який момент часу, шляхом застосування WAL до існуючої резервної копії.

– **Реплікація та технологія Hot Standby.** Починаючи з версії 9.0, на основі WAL введена реплікація за технологією Hot Standby. Технологія дозволяє отримати на сервері другу базу даних, яка є актуальною копією оригінальної бази даних, доступної лише для читання. Технологія може бути використана також і на віддаленому сервері, який підключається до primary- або master-сервера і завантажує з нього WAL-логи, надаючи онлайнову реплікацію бази даних і підтримуючи копію бази даних на віддаленому сервері в актуальному стані, а також роблячи цю копію доступною для запитів на читання.

– **Налаштування.** Таблиці можуть успадковувати характеристики і набори полів від інших таблиць (батьківських). При цьому дані, які додаються до породженої таблиці, автоматично будуть брати участь (якщо це не зазначено окремо) в запитах до батьківської таблиці. Ця функція в даний час не є повністю завершеною, проте її стан досить для практичного використання.

– **Гнучке налаштування серверу.** Основний конфігураційний файл postgresql.conf включає більше 150 параметрів, що настроюються по розділах: файли і шляхи до них, авторизація та безпеку, виділення ресурсів і т.д. Додатковий конфігураційний файл pg_hba.conf включає в себе налаштування доступу до окремих БД, такі як вказівку конкретних IP-адрес і (або) мереж, з яких дозволений доступ, а також метод авторизації для доступу в БД і можливість включення безпечних (зашифрованих) з'єднань.

Фізична архітектура СУБД PostgreSQL

Всі дані, необхідні для кластера бази даних зберігаються в каталозі, який, як правило, називають PGDATA (як і відповідну змінну оточення). PGDATA зазвичай локалізована в ../var/lib/pgsql/data. Кілька кластерів, керованих різними екземплярами сервера, можуть існувати на одній машині.

Каталог PGDATA містить кілька підкаталогів і файлів управління, таких як: PG_VERSION – файл, який містить номер версії PostgreSQL, base – підкаталог, що містить підкаталоги баз даних, і ін., А також необхідні файли конфігурації кластера postgresql.conf, pg_hba.conf і pg_ident. conf (хоча в PostgreSQL 8.0 і вище, можна зберегти їх в іншому місці).

Для кожної бази даних в кластері є підкаталог PGDATA / base, ім'я якого збігається з OID (object identifier – ідентифікатор об'єкта) бази даних, що зберігаються в системній таблиці pg_database. Цей підкаталог використовується для зберігання файлів бази даних, зокрема, його системних каталогів.

Кожна таблиця і індекс зберігається в окремому файлі з ім'ям, що збігається з дескриптором таблиці (filenode). Крім того кожна таблиця або індекс має карту вільного простору (free space map), в якій зберігається інформація про наявний обсяг пам'яті. Карта вільного простору зберігається в файлі з ім'ям, що складається з десктриптору таблиці або індексу і суфікса _fsm.

PostgreSQL підтримує **табличні простори** (**tablespaces**), які дозволяють задати місце зберігання об'єктів БД в файлової системі. Спочатку створюється табличний простір з певним

ім'ям. Далі, це ім'я може бути використано при створенні таблиць, щоб розмістити ці таблиці саме в даному табличному просторі. Кожний визначений користувачем табличний простір має посилання всередині каталогу PGDATA/pg_tblspc, яке вказує на фізичний каталог цього табличного простору. Це посилання має таке ж ім'я, як і OID табличного простору. Усередині фізичного каталогу табличного простору є підкаталог з ім'ям, яке залежить від версії PostgreSQL сервера, наприклад, PG_9.0_201008051. У цьому підкаталозі містяться підкаталоги для кожної бази даних (їх імена збігаються з OID бази даних), яка має елементи в даному табличному просторі. Для іменування записи таблиць і індексів в цих підкаталогах використовується схема, заснована на filenode.

Тимчасові файли (для таких операцій, як сортування великих обсягів даних) створюються в PGDATA /base/pgsql_tmp, або в підкаталозі pgsql_tmp каталогу табличного простору, якщо для них зазначено табличний простір, відмінне від табличного простору за замовчуванням. Ім'я тимчасового файлу має вигляд pgsql_tmpPPP.NNN, де PPP – OID обчислювальної машини бази даних (backend), NNN – мітки різних тимчасових файлів.

Логічна архітектура СУБД PostgreSQL

До особливостей логічної архітектури PostgreSQL, крім згадуваних вище табличних просторів, можна віднести наступні моменти:

– **Схеми**

PostgreQL підтримує схеми. Схеми є як би додатковими областями видимості всередині бази данних. Також схему можна порівняти і з додатковим шляхом (назва схеми має відзначатися перед назвою таблиці) і з каталогом, всередині якого можна розмістити таблиці. У будь-якій базі даних за замовчуванням існує схема public, в якій за замовчуванням створюються всі таблиці і яку не потрібно вказувати спеціально. Але адміністратор БД може створювати інші схеми (і розмежовувати доступ до них), що забезпечує ще один рівень розподілу прав доступу для користувачів, дозволяє виділити кожному користувачеві як би персональний розділ всередині БД з тими ж назвами таблиць, що і у інших користувачів.

– **Індекси**

POSTGRESQL пропонує 4 типи індексів: B-tree, Hash, GiST (Generalized Search Tree) і GIN (Generalized Inverted Index). Кожен тип індексу має свій алгоритм реалізації, що дозволяє істотно збільшити швидкодію, якщо для певного виду даних вибирати певний тип індексу.

POSTGRESQL дозволяє також створювати індекси з використанням виразів і часткові (partial) індекси (з використанням службового слова WHERE).

– **Ролі та привілеї**

PostgreSQL управляє привілеями в БД, використовуючи концепцію ролей. Роллю може бути як окремий користувач БД, так і група користувачів. Ролі можуть бути власниками об'єктів в БД (наприклад таблиць), а також можуть призначати привілеї доступу до цих об'єктів для інших ролей. Можливо надати одній ролі членство в іншій ролі і відповідно передати цій ролі права тієї ролі, членом якої вона буде. Концепція ролей замінила стару концепцію користувачів і груп, надавши ту ж функціональність. Починаючи з версиии 9.0 в PostgreSQL підтримуються права на схеми і права за замовчуванням.

– **Правила**

Механізм правил (rules) являє собою механізм створення користувацьких обробників не тільки операцій маніпулювання, а й операцій вибірки даних. Основна відмінність від механізму тригерів полягає в тому, що правила спрацьовують на етапі розбору запиту, до вибору оптимального плану виконання і самого процесу виконання. Правила дозволяють перевизначати поведінку системи при виконанні SQL-операцій до таблиці. Наприклад, при створенні представлень (views) створюється правило, яке визначає, що замість виконання операції вибірки до представлення система повинна виконувати операцію вибірки до базової таблиці/таблиць з урахуванням умов вибірки, які лежать в основі визначення представлення. Для створення представлень, які підтримують операції оновлення, правила для операцій вставки, зміни та видалення рядків, повинні бути визначені користувачем.

Система правил (більш правильно говорити: система правил зміни запитів) дозволяє змінювати запит згідно із заданими правилами і потім передає змінений запит планувальником запитів для планування і виконання. Система правил є дуже потужним інструментом і може бути використана в багатьох випадках, таких як збережені процедури і представлення.

– **Збережені процедури і тригери**

Збережені процедури в PostgreSQL можуть бути написані на будь-якій з підтримуваних вбудованих мов. Збережені процедури можуть бути використані в тригерах і можуть повертати будь-який з підтримуваних типів даних, а також масиви і списки. Починаючи з версії 9.0, викликати збережені процедури можна із зазначенням іменованих параметрів.

Тригери визначаються як функції, які ініціюються операціями маніпулювання. Тригери можуть бути призначені до або після операцій INSERT, UPDATE або DELETE. Якщо відбулася подія, на яку був призначений тригер, то викликається закріплена за цим тригером процедура. Наприклад, операція INSERT може запускати тригер, перевіряючий доданий запис на відповідність певним умовам. При написанні функцій для тригерів можуть використовуватися різні мови програмування. Тригери асоціюються з таблицями і виконуються в алфавітному порядку.

У версії 9.0 введені тригери на стовпці і, крім того, при оголошенні тригера можна використовувати ключове слово WHEN, яке додає додаткову умову для спрацьовування тригера.

– **Функції**

Функції є блоками коду, що виконуються на сервері, а не на боці клієнта БД. Іноді функції ототожнюються зі збереженими процедурами, проте між цими поняттями є різниця. Хоча вони можуть створюватися на чистому SQL, реалізація додаткової логіки, наприклад, умовних переходів і циклів, виходить за рамки власне SQL і вимагає використання деяких мовних розширень. Функції можуть писатися на одній з таких мов:

– Вбудована процедурна мова PL/pgSQL, яка багато в чому аналогічна мові PL/SQL, що використовується в СУБД Oracle;

– Скриптові мови – PL/Lua, PL/LOLCODE, PL/Perl, PL/PHP, PL/Python, PL/Ruby, PL/sh, PL/Tcl та PL/Scheme ;

– Класичні мови – C, C ++, Java (через модуль PL/Java);

– Статистична мова R (через модуль PL/R).

PostgreSQL допускає використання функцій, які повертають набір записів, який надалі можна використовувати так само, як і результат виконання звичайного запиту (курсор). Функції можуть виконуватися як з правами їх творця, так і з правами поточного користувача.

Починаючи з 9.0, можна створювати функції без оголошення імені (анонімні блоки) для виконання блоку операторів на будь-якій вбудованій мові, яку підтримує PostgreSQL, прямо в командному рядку.

– **Типи даних**

PostgreSQL підтримує великий набір вбудованих типів даних:

1) Числові типи
   - Цілі
   - З фіксованою точкою
   - С плаваючею точкою
   - Грошовий (відрізняється спеціальним форматом виведення, а в іншому аналогічний числам з фіксованою точкою і двома знаками після коми)
2) Символьні типи довільної довжини
3) Двійкові типи (включаючи великий двійковий об'єкт BLOB – Binary Large Object)
4) Типи «дата/час» (повністю підтримують різні формати, точність, формати виведення, включаючи останні зміни в часових поясах)
5) Логічний тип
6) Перелічувальний тип

7) Геометричні примітиви
8) Мережні типи
   - IP і IPv6-адреси
   - CIDR-формат
   - CIDR (Classless Inter-Domain Routing) – безкласова адресація – метод IP-адресації, що дозволяє гнучко управляти простором IP-адрес, не використовуючи жорстких рамок IP-адресації на основі класів мереж.
   - MAC-адреси
   - Унікальний ідентифікатор, що присвоюється кожній одиниці мережевого обладнання.
9) UUID тип

UUID (Universally Unique Identifier) – це стандарт ідентифікації, що використовується при створенні програмного забезпечення. Найбільш поширеним використанням даного стандарту є Globally Unique Identifier (GUID) фірми Microsoft.

10) XML тип

XML (Xtensible Markup Language) – текстовий формат, призначений для зберігання структурованих даних (замість існуючих файлів баз даних), для обміну інформацією між програмами, а також для створення на його основі більш спеціалізованих мов розмітки.

11) Масиви
12) OID-типи

OID (Object identifiers) представляють ідентифікатори різних об'єктів і використовуються зазвичай в PostgreSQL як первинні ключі для різних системних таблиць. Ці типи представляються як 4-байтові цілі числа без знака, тобто мають досить обмежений діапазон значень, тому не можуть використовуватися у великих базах даних.

13) Композитні типи

Композитний (складений) тип (composite type) представляє структуру ряду або запису, тобто по суті список імен атрибутів та їх типів даних.

14) Псевдотипи

Псевдотипи (pseudo-types) не можуть використовуватися в якості типу даних стовпця таблиці або подання, але можуть використовуватися для оголошення аргументів функції або типу результату.

З будь-яким об'єктом даних, представлених в PostgreSQL, пов'язують певний тип, навіть якщо на перший погляд це і не очевидно. Тип даних одночасно визначає і обмежує різновиди операцій, які можуть виконуватися з цими даними.

Хоча більшість типів даних PostgreSQL взято безпосередньо зі стандартів SQL, існують і інші, нестандартні типи даних (наприклад, геометричні та мережеві типи). У табл. 2.1 перераховані основні базові типи даних PostgreSQL, а також їх синоніми (альтернативні імена).

Таблиця 2.1 – Типи даних PostgreSQL

Тип даних	**Опис**	**Стандарт**
Логічні та двійкові типи даних		
boolean, bool	Окрема логічна величина (true або false)	SQL99
bit(n)	Бітова послідовність фіксованої довжини (рівно n біт)	SQL92
bit varying(n),varbit(n)	Бітова послідовність змінної довжини (до n біт)	SQL92
Символьні типи		
character(n), char(n)	Символьна строка фіксованої довжини (рівно n символів)	SQL89

Продовження таблиці 2.1

Тип даних	Опис	Стандарт
text	Символьна строка змінної або необмеженої довжини	PostgreSQL
character varying(n), varchar(n)	Символьна строка змінної довжини (до n символів)	SQL92
Числові типи		
small int, int2	2-байтове ціле зі знаком	SQL89
integer, int, int4	4-байтове ціле зі знаком	SQL92
bigint, int8	8-байтове ціле зі знаком, до 18 цифр	PostgreSQL
real, float4	4-байтове дійсне число	SQL89
double precision, floats, float	8-байтове дійсне число	SQL89
numeric(p.s), decimal (p.s)	Число з p цифр, що містить 5 цифр в дробовій частині	SQL99
money	Фіксована точність, подання грошових величин	PostgreSQL, вважається застарілим
serial	4-байтове ціле з автоматичним збільшенням	PostgreSQL
Час та дата		
date	Календарна дата (день, місяць та рік)	SQL92
time	Час доби	SQL92
time with time zone	Час доби з інформацією про часовий пояс	SQL92
timestamp	Дата та час	SQL92
interval	Довільний інтервал часу	SQL92
Геометричні типи		
box	Прямокутник на площині	PostgreSQL
line	Нескінченна лінія на площині	PostgreSQL
Iseg	Відрізок на площині	PostgreSQL
circle	Коло з заданим центром і радіусом	PostgreSQL
path	Замкнена або розімкнена геометрична фігура на площині	PostgreSQL
point	Точка на площині	PostgreSQL
polygon	Замкнений багатокутник на площині	PostgreSQL
Мережні типи		
cidr	Специфікація мережі IP	PostgreSQL
inet	Мережний IP-адрес з необов’язковими бітами підмережі	PostgreSQL

Продовження таблиці 2.1

Тип даних	Опис	Стандарт
macaddr	MAC-адрес (наприклад, апаратна адреса адаптеру Ethernet)	PostgreSQL
Системні типи		
oid	Ідентифікатор об’єкту (запису)	PostgreSQL
xid	Ідентифікатор транзакції	PostgreSQL

Крім вбудованих типів даних, користувач може самостійно створювати нові необхідні йому типи і програмувати для них механізми індексування с допомогою методів GiST.

Призначені для користувача об'єкти

PostgreSQL може бути розширений користувачем для власних потреб практично в будь-якому аспекті. Є можливість додавати:

- Типи даних та їх перетворення.
- Домени.
- Функції (включаючи агрегатні).
- Індекси.
- Оператори (включаючи перевизначення вже існуючих).
- Процедурні мови.

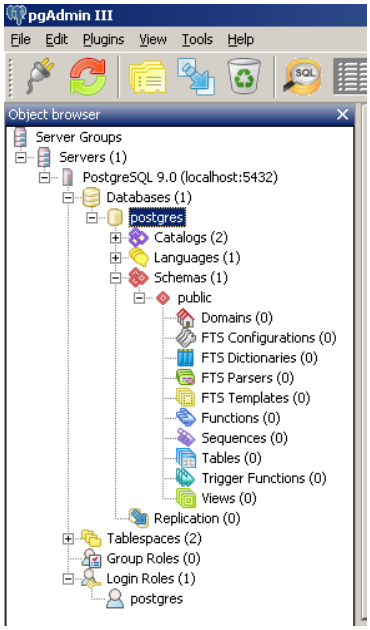

Рисунок 2.1 – Основні компоненти бази даних PostgreSQL 9.0.

Як і в SQL Server, дані в базі даних PostgreSQL організовані в декількох різних об'єктах (рис. 2.1):

- домени (domains);
- конфігурація повнотекстового пошуку (FTS configuration);
- словники повнотекстового пошуку (FTS dictionaries);
- синтаксичні аналізаори повнотекстового пошуку (FTS parsers);
- шаблони повнотекстового пошуку (FTS templates);
- функції (functions);
- послідовності (sequences);
- таблиці (tables);
- тригери (trigger functions);

–представлення (views).

2.1.2 Розробка логічної моделі бази даних

2.1.2.1 Засоби для розробки та адміністрування баз даних

Для роботи з базами даних існує декілька можливостей:

– Запуск інтерактивної термінальної програми, яка дозволяє вводити, редагувати і виконувати команди SQL:

СУБД	Інтерфейс командної строки
MS SQL Server	Isqlw
PostgreSQL	Psql

– Використання пакету з графічним інтерфейсом (GUI):

СУБД	GUI
MS SQL Server	SQL Server Enterprise Manager
PostgreSQL	pgAdmin

– Написання спеціального додатку, використовуючи один з декількох доступних мов програмування, які підтримуються СУБД.

Далі буде розглянута робота в середовищі PostgreSQL з використанням pgAdmin.

2.1.2.2 Приклад створення бази даних

Крок 1: Створення бази даних BookShop

Параметри нової бази даних показані на рис. 2.2.

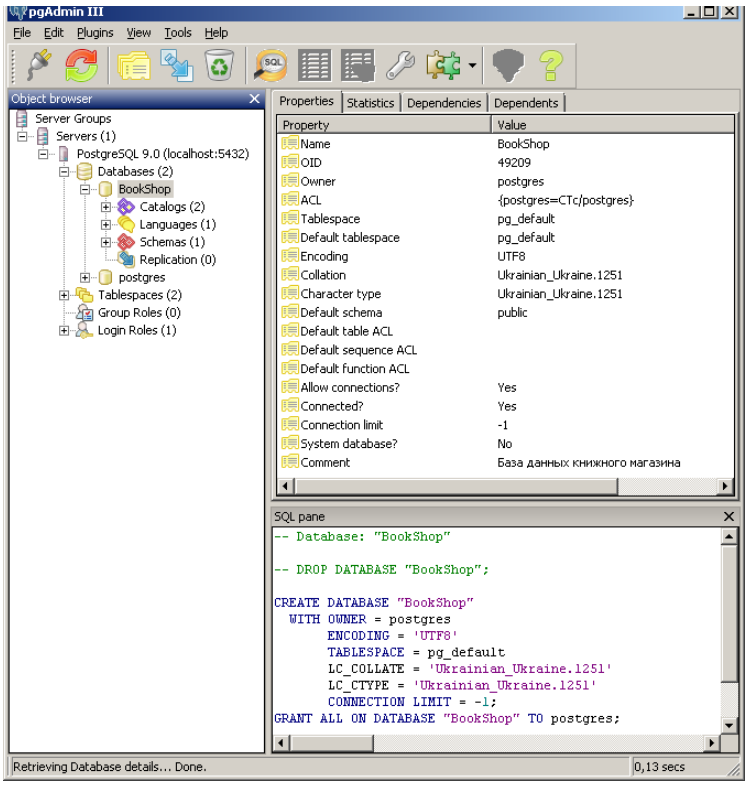

Рисунок 2.2 – Параметри бази даних BookShop

При створенні нової бази даних в системну таблицю pg_database (див. Pg_catalog/Tables) додається рядок, що містить параметри цієї бази даних.

Крок 2: Створюємо таблиці Книги, Постачальники, Замовники, Замовлення, Поставки.

Після того, як база даних створена, можна приступати до створення її основних об'єктів

– таблиць. Перш за все, для кожного стовпця будь-якої таблиці необхідно вказати певний тип даних (табл. 2.2). У SQL тип даних задається після імені стовпчика за допомогою відповідного ключового слова, потім вводяться параметри представлення значень стовпців, такі як довжина, значення за замовчуванням та ін. Після визначення тип даних стовпця таблиці зберігається у вигляді постійної характеристики стовпчика і не може бути змінений.

Відносно типів даних слід пам'ятати, що неприпустимо називати об'єкти іменами команд або використовувати для цієї мети інші зарезервовані слова. Типи даних – це повноцінні об'єкти бази даних, які зберігаються в системній таблиці pg_type разом з їх OID.

Таблиця 2.2 – Перелік таблиць з визначеними полями та їх типами

Таблиця	**Поле**	**Тип**
Книги	Код_книги	serial
	Автор	character varying(80)
	Назва	character varying(160)
	Видавництво	character varying(80)
	Ціна	money
	Залишок	smallint
Постачальники	Код_постачальника	serial
	Назва	character varying(40)
	Місто	character varying(40)
	Адреса	character varying(80)
	Телефон	character(13)
Замовлення	Код_замовлення	serial
	Код_книги	integer
	Код_замовника	integer
	Сплачено	character varying(3)
	Дата	date
Замовники	Код_замовника	serial
	Ім’я	character varying(40)
	Адреса	character varying(80)
	Телефон	character(13)
Поставки	Номер	serial
	Код_книги	integer
	Код_постачальника	integer
	Кількість	integer
	Дата	date

Крок 3: Введення обмежень цілісності.

Наступний момент в процесі створення таблиць, якому необхідно приділити особливу увагу, пов'язаний із забезпеченням цілісності даних. Для забезпечення цілісності даних в таблицях визначаються обмеження на значення стовпців (constraints). Ці обмеження можуть бути введені при створенні таблиці для кожного стовпця окремо або додані в таблицю пізніше за допомогою спеціальної команди SQL ALTER TABLE. У PostgreSQL підтримуються наступні основні обмеження цілісності:

- PRIMARY KEY – первинний ключ.
- FOREIGN KEY/REFERENCES – зовнішній ключ (посилання).
- UNIQUE – унікальність.
- CHECK – перевірка умови на значення.

Обмеження первинного ключа на значення стовпця використовується для забезпечення унікальності даних в стовпчиках та в цілому для забезпечення посилальної цілісності (при зв'язуванні таблиць за допомогою зовнішніх ключів). Визначення умови primary key для таблиці має кілька ефектів. По-перше, воно встановлює певні умови на

значення первинного ключа – забороняється введення однакових значень та значень NULL в ті стовпці, для яких воно визначено. По-друге, primary key створює унікальний індекс для цих стовпців, що дозволяє прискорити пошук рядків в таблиці.

Визначення умови primary key в одній таблиці саме по собі не забезпечує цілісність по посиланнях. Необхідно також визначити відповідні зовнішні ключі тих таблиць, рядки яких будуть комбінуватися з рядками тієї таблиці, де визначено обмеження на значення колонки PRIMARY KEY.

Обмеження зовнішнього ключа на значення стовпця зазвичай застосовується разом з попередньо визначеним обмеженням primary key (насправді досить обмеження UNIQUE) в асоційованій таблиці. Умова на значення foreign key ставить у відповідність один або декілька стовпців таблиці ідентичному набору стовпців іншої таблиці, для яких визначено обмеження primary key (або UNIQUE). Коли оновлюються або видаляються значення тих стовпців таблиці, на які посилаються зовнішні ключі інших таблиць, виникає питання: що робити з відповідними значеннями ключів? Існує кілька варіантів вирішення цієї проблеми:

– нічого не робити (**no action**) (ця подія має оброблятися деяким відмінним від стандартного способом, інакше буде видаватися повідомлення про помилку);

– заборонити будь-які зміни (**restrict**);

– автоматично оновити / видалити значення відповідних зовнішніх ключів (**cascade**),

– встановити для зовнішніх ключів NULL-значення (**set null**) (для цього відповідні стовпці не повинні мати обмеження NOT NULL);

– встановити для зовнішніх ключів значення за замовчуванням (**set default**).

Автоматичне оновлення відповідних стовпців в різних таблицях після того, як для них визначені обмеження на значення стовпців primary key та foreign key, називається декларативною посилальної цілісністю (declarative referential integrity).

Обмеження на значення стовпців primary key та foreign key забезпечують відповідність рядків пов'язаних таблиць, тому стовпці з такими обмеженнями використовуються для реалізації операції з'єднання таблиць.

Наочне уявлення про структуру зв'язків між таблицями в базі даних можна отримати за допомогою діаграм «таблиця-зв'язок», на яких вказуються обмеження primary key й foreign key та така характеристика зв'язків, як ступінь зв'язку. На рис. 2.3 показана діаграма бази даних BookShop.

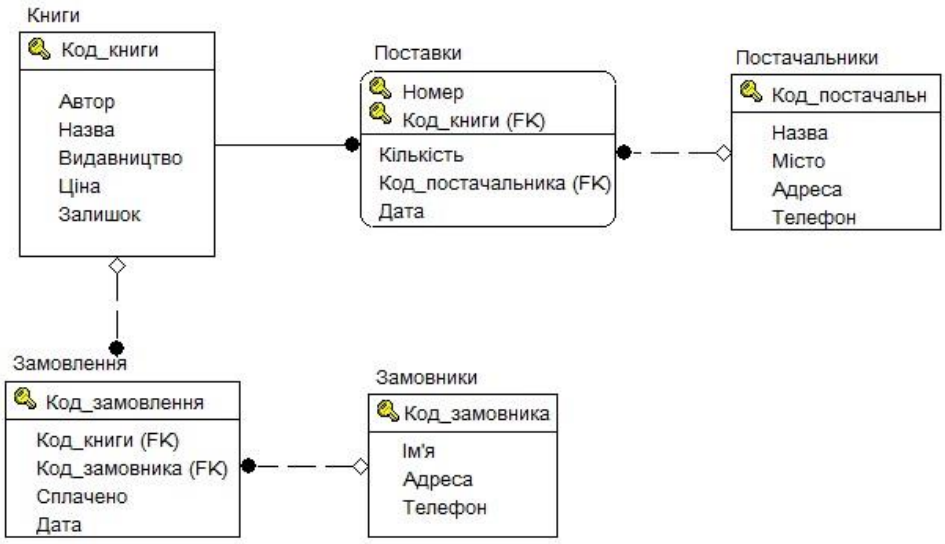

Рисунок 2.3 – Діаграма «таблиця-зв'язок» бази даних BookShop. Суцільна лінія – які ідентифікують зв'язок, пунктирні – які не ідентифікують зв'язок. Всі зв'язки мають ступінь 1: N (N – з боку залежної таблиці).

Обмеження унікальності на значення стовпця можна призначити для того, щоб заборонити повторення значень в будь-якому стовпці таблиці. Для стовпчика, що входить до

складу первинного ключа, подібне обмеження не може бути визначено, тому що обмеження унікальності реалізується за допомогою автоматичного створення унікального індексу для стовпця таблиці, а для первинного ключа також автоматично створюється унікальний індекс. Однак, стовпець (або стовпці, якщо обмеження UNIQUE визначено відразу для декількох стовпців) з обмеженням UNIQUE може бути використаний для зв'язку з іншими таблицями, тобто на нього можуть посилатися зовнішні ключі цих таблиць.

Перевірочне обмеження на значення стовпця встановлює діапазон значень, які можуть бути введені в один або декілька стовпців таблиці бази даних. Обмеження check може використовуватися, наприклад, для того, щоб встановити діапазон значень, які допускається зберігати в стовпці, визначеному для даних числових типів.

Процес установки перевірки значення для стовпця таблиці називається зв'язуванням (binding). Можна визначити й ввести кілька перевірок в один стовпець. Перевірка може бути визначена для стовпця навіть в тому випадку, якщо для нього вже існує якесь правило.

Хоча перевірочні обмеження на значення працюють швидше й встановлюються простіше, але правила гнучкіше. Після визначення правила воно може бути пов'язане зі стовпцями декількох таблиць, але для одного стовпця можна задати лише одне правило й кілька перевірочних умов.

Далі вводимо для стовпців створених раніше таблиць наступні обмеження (табл. 2.3).

Таблиця 2.3 – Обмеження для стовпців таблиць

Таблиця	**Поле**	**Обмеження**
Постачальники	Код_постачальника	Primery key
	Назва	Unique
	Телефон	Defaults (“000 111-11-11”)
Замовлення	Код_замовлення	Primery key
	Код_книги	Foreign key (Код_книги із Книги, NULL-значення не допускаються, автоматичне оновлення при зміні Код_книги із Книги, видалення із Книги не допускаються).
	Код_замовника	Foreign key (Код_замовника із Замовники, NULL-значення не допускаються, автоматичне оновлення при зміні Код_замовника із Замовники, видалення із Замовники не допускаються)
	Сплачено	Check (“Так”, “Ні”)
	Дата	Defaults (timenow())
Замовники	Код_замовника	Primery key
	Телефон	Defaults (“000 111-11-11”)
Книги	Код_книги	Primery key
Поставки	Номер	Primery key
	Код_книги	Primery key Foreign key (Код_книги із Книги, NULL-значення не допускаються, оновлення при зміні Код_книги із Книги не допускаються, видалення із Книги не допускаються).
	Код_постачальника	Foreign key (Код_постачальника із Постачальники, NULL-значення допускаються, автоматичне оновлення при зміні Код_постачальника із Постачальники, при видаленні із Постачальники встановити в NULL)

Насамкінець, для кожної таблиці генеруємо звіт Data Dictionary Report. Для таблиці Поставки, наприклад, він повинен виглядати так:

Table Data dictionary report - Поставки
Generated: 25.09.2016 19:49:30
Server: PostgreSQL 9.3 (localhost:5432)
Database: BookShop
Schema: public
Columns

Name	Data type	Not Null?	Primary key?	Default
Номер	integer	Yes	Yes	nextval('"Поставки_Номер_seq"'::regclass)
Код_книги	integer	Yes	No	
Код_поставщика	integer	No	No	
Количество	integer	No	No	
Дата	date	No	No	

Constraints

Name	Type	Definition
pk_поставки	Primary key	("Номер")
fk_поставки_1	Foreign key	("Код_книги") REFERENCES "Книги" ("Код_книги") MATCH SIMPLE ON UPDATE RESTRICT ON DELETE RESTRICT
fk_поставки_2	Foreign key	("Код_постачальника") REFERENCES "Постачальник" ("Код_постачальника") MATCH SIMPLE ON UPDATE CASCADE ON DELETE SET NULL

Крок 4: Введення даних в таблиці.

Для перегляду і маніпулювання даними в таблицях в pgAdmin призначені відповідні команди з меню Tools або кнопки на панелі інструментів. При введенні даних в таблиці слід пам'ятати про обмеження, які були визначені для стовпців і таблиць бази даних. Не слід забувати, що: 1) зовнішні ключі можуть приймати тільки ті значення, які вже введені (!) В поля тих таблиць, на які ці зовнішні ключі посилаються; 2) в стовпці з типом serial (лічильник) не можна самим вводити значення; 3) якщо в стовпець, для якого визначено значення за замовчуванням, не водиться жодного значення, то автоматично буде введено значення за замовчуванням.

Для того, щоб ввести в таблицю нове значення, необхідно просто перейти до рядка, зазначеної зірочкою (*) і почати вводити необхідні значення у відповідні стовпці.

Після того, як введені всі нові рядки або після зміни даних, необхідно зафіксувати цей факт в базі даних, для чого потрібно виконати команду Refresh, натиснувши однойменну кнопку на панелі інструментів вікна Edit Data.

Щоб видалити який-небудь рядок таблиці, досить виділити її, клацнув лівою кнопкою миші по її номеру, і виконати команду Delete контекстного меню. Але при цьому слід пам'ятати про цілісність даних по посиланнях, якщо вона застосовується в базі даних.

Вводимо кілька довільних рядків в таблиці бази даних BookShop з урахуванням певних обмежень на значення стовпців.

2.2 Контрольне завдання

2.2.1. На основі інфологічної моделі, розробленої в лабораторій роботі №1 створити БД та ввести тестові дані в кожну з створених таблиць.

2.2.2 Оформити звіт по результатам виконання контрольного завдання.

Звіт повинен містити:

1) Інфологічну модель БД (на мові таблиця-зв’язок).

2) Розроблені обмеження цілісності.

3) Звіти Data Dictionary Report, сгенеровані для кожної створеної при виконанні контрольного завдання таблиці.

4) Звіти Data Dictionary Report, сгенеровані для кожної таблиці БД BookShop.

2.3 Контрольні питання

1) Сформулюйте основні властивості реляційної таблиці.
2) Що таке первинний ключ?
3) З яких файлів складається база даних SQL Server?
4) Сформулюйте правила забезпечення цілісності даних в СУБД.
5) Що таке інфологічна модель бази даних?
6) Що таке логічна модель бази даних?
7) З яких стандартних об'єктів складається модель бази даних SQL Server?
8) Що таке транзакція?
9) Яка структура файлу бази даних SQL Server?
10) Для чого і як визначаються обмеження на значення стовпців?
11) Що таке обмеження первинного ключа?
12) Що таке обмеження зовнішнього ключа?
13) Що таке зв'язування?
14) Для чого призначені правила? У чому різниця між правилами і обмеженнями на значення стовпців?
15) У чому різниця між значенням за замовчуванням і установкою за замовчуванням?
16) Що таке представлення?
17) Що таке тригер?
18) Що таке обмеження цілісності? Які обмеження цілісності підтримує PostgreSQL?
19) Для чого призначена і в чому суть технології MVCC?
20) Що таке WAL?
21) Яка структура сторінок файлу даних PostgreSQL?
22) У чому полягає концепція табличних просторів, яка використовується в PostgreSQL?
23) Що таке схеми в PostgreSQL? Для чого вони призначені?

Лабораторна робота №3

SQL: ВИБІРКА ДАНИХ

Знайомство з командами вибірки даних мови plpgsql. Синтаксис інструкції SELECT. Вибірка без використання фрази WHERE. Вибірка с використанням фрази WHERE. Вибірка з упорядкуванням. Вибірка і агрегування даних. Перетворення типів даних в інструкції SELECT. Використання SELECT для з'єднання двох і більше таблиць. Вкладені підзапити. Реалізація операцій реляційної алгебри за допомогою фрази SELECT.

3.1 Теоретичні відомості

3.1.1 Знайомство з Query Tool pgAdmin

Для введення і тестування інструкцій і операторів мови plpgsql будемо використовувати додаток Query Tool, що входить в комплект поставки PosgreSQL. Для запуску Query Tool необхідно ввести однойменну команду з меню Tools командного меню pgAdmin. У вікні (рис. 3.1) вибрати базу даних зі списку та створити новий запит або завантажити збережений раніше SQL-скрипт.

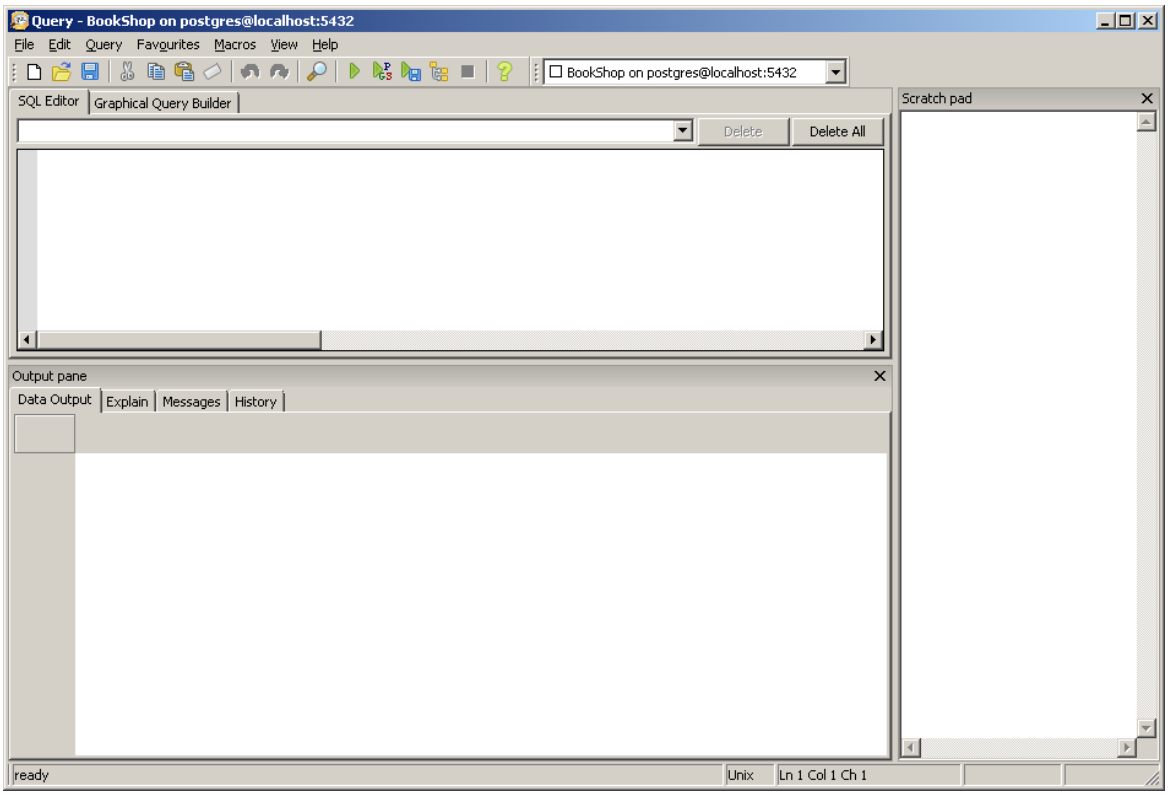

Рисунок 3.1 – Основне вікно Query Tool

Query Tool – багатовіконний графічний засіб, який дозволяє:

– вводити (New query, Load SQL script), редагувати, виконувати (Execute query), та зберігати (Save query/result) запити та їх результати в текстовому (text) та графічному вигляді (image);

– візуально будувати запит (Graphical Query Builder);

– отримувати пояснення, як запити аналізуються, оптимізуються і виконуються, і представляти в графічному вигляді схему виконання запиту (Explain query);

– отримувати статистичну інформацію про запит (Message);

– зберігати результати виконання запитуа в файл (Export).

3.1.2 Синтаксис інструкції SELECT

Всі запити на отримання практично будь-якої кількості даних з однієї або декількох таблиць виконуються за допомогою єдиного фрази SELECT. У загальному випадку результатом реалізації фрази SELECT є інша таблиця. До цієї нової (робочої) таблиці може бути знову застосована операція SELECT і т.д., тобто такі операції можуть бути вкладені одна в одну. Являє історичний інтерес той факт, що саме можливість включення одної фрази SELECT всередину іншого послужила мотивуванням використання прикметника "структурований" в назві мови SQL.

Інструкція SELECT використовується в основному як:

– самостійна команда на отримання та виведення рядків таблиці, сформованої з стовпців і рядків однієї або декількох таблиць (представлень);

– елемент іншого запиту, так званий «вкладений запит»;

– фраза вибору в командах створення представлення, курсору або вставки;

– засіб присвоєння змінним значень з рядків сформованої таблиці.

У цій роботі розглядаються тільки два перші варіанти використання SELECT і його синтаксис, обмежений конструкціями, що використовуються при реалізації цих варіантів, а решта – в наступних роботах практикуму.

Тут і далі в синтаксичних конструкціях можуть використовуватися такі символи:

– зірочка (*) для позначення "все" – вживається в звичайному для програмування сенсі, тобто "Всі випадки, що задовольняють визначенню";

– квадратні дужки ([]) – означають, що конструкції, укладені в ці дужки, є необов'язковими (тобто можуть бути опущені);

– фігурні дужки ({}) – означають, що конструкції, укладені в ці дужки, повинні розглядатися як цілі синтаксичні одиниці, тобто вони дозволяють уточнити порядок розбору синтаксичних конструкцій, замінюючи звичайні дужки, використовувані в синтаксисі SQL;

– три крапки (...) – вказує на те, що безпосередньо передує йому синтаксична одиниця може повторюватися один або більше разів;

– пряма риска (|) – означає наявність вибору з двох або більше можливостей. Наприклад, позначення ASC | DESC вказує, можна вибрати один з термінів ASC або DESC; коли ж один з елементів вибору укладений у квадратні дужки, то це означає, що він вибирається за замовчуванням (так, [ASC] | DESC означає, що відсутність всієї цієї конструкції буде сприйматися як вибір ASC);

– крапка з комою (;) – завершальний елемент пропозицій SQL;

– кома (,) – використовується для поділу елементів списків;

– прогалини ( ) – можуть вводитися для підвищення наочності між будь-якими синтаксичними конструкціями пропозицій SQL;

– великі латинські букви і символи – використовуються для написання конструкцій мови SQL і повинні (якщо це спеціально не обумовлено) записуватися в точності так, як показано;

– малі літери – використовуються для написання конструкцій, які повинні замінюватися конкретними значеннями, обраними користувачем, причому для визначеності окремі слова цих конструкцій зв'язуються між собою символом підкреслення (_);

– терміни таблиця, стовпець, ... - заміняють (з метою скорочення тексту синтаксичних конструкцій) терміни ім'я_таблиці, ім'я_стовпця, ..., відповідно;

– термін таблиця використовується для узагальнення таких видів таблиць, як базова_таблиця, представлення або псевдонім; тут псевдонім служить для тимчасового (на момент виконання запиту) перейменування і (або) створення робочої копії базової таблиці або подання.

Інструкція SELECT (вибрати) має наступний формат (спрощений):

*SELECT [[ALL] | DISTINCT] { * | елемент_SELECT [[AS] [псевдонім_стовпця]*
[,елемент_SELECT [AS] [псевдонім_стовпця]] ...}
FROM {базова_таблиця | представлення} [AS] [псевдонім]
[,{базова_таблиця | представлення } [AS] [псевдонім]] ...
[WHERE умова_where]
[GROUP BY список_ group_by
[HAVING умова_having]]
[ORDER BY список_order_by
[FOR UPDATE [OF таблиця [. ...]]] [LIMIT { число | ALL } [{ OFFSET | . } початок]]

Різні елементи цієї інструкції дозволяють вказати умови для вибору потрібних даних і (за потреби) їх обробки:

SELECT (вибрати) дані із зазначених стовпців і (якщо необхідно) виконати перед виведенням їх перетворення відповідно до зазначених виразів і (або) функцій;

FROM (із) перерахованих таблиць, в яких розташовані ці стовпці;

WHERE (де) рядки із зазначених таблиць повинні задовольняти зазначеному переліку умов відбору рядків;

GROUP BY (групуючи по) зазначеному переліку стовпців з тим, щоб отримати для кожної групи єдине агреговане значення, використовуючи в списку елементів SELECT агрегуючі SQL-функції, наприклад: SUM (сума), COUNT (кількість), MIN (мінімальне значення), MAX (максимальне значення) або AVG (середнє значення);

HAVING (маючи) в результаті лише ті групи, які задовольняють зазначеному переліку умов відбору груп;

ORDER BY (сортуючи) результати вибору даних; при цьому впорядкування можна виробляти в порядку зростання – ASC (ASCending) або зменшення – DESC (DESCending), а за замовчуванням приймається ASC.

Синтаксичні конструкції, введені вище, визначаються в такий спосіб:

Елемент_SELECT ::= [таблиця.] | вираз*
вираз ::= константа
| скалярна_функція
| [псевдонім.]стовпець
| (вираз)
| (скалярний_підзапит)
| {унарний оператор} вираз
| вираз {бінарний_оператор} вираз

де:

константа – одиничні алфавітно-цифрові символи, рядки символів і числові константи (символьні дані і дані типу дата/час беруться в одинарні лапки);

скалярна_функція – SQL-функція, яка повертає скалярне значення, наприклад SUM, CAST та ін. (вирази, що виступають в якості параметрів SQL-функцій, не повинні містити вкладених SQL-функцій).

базова_скалярна_функція ::=
| {SUM|AVG|MIN|MAX|COUNT} ([[ALL]|DISTINCT][таблиця.]стовпець)
| {SUM|AVG|MIN|MAX|COUNT} ([ALL] вираз)
| COUNT(*)

де:

псевдонім – тимчасовий ідентифікатор, який присвоюється таблиці за допомогою службового слова AS в реченні FROM;

стовпець – власне ім'я стовпця, що складається з імені бази даних, власника, таблиці і імені стовпця;

скалярний_підзапит – підзапит, повертає скалярний результат; підзапит – це інструкція SELECT, вкладена в:

– фрази WHERE, HAVING або SELECT іншої інструкції SELECT;

– інструкції INSERT, UPDATE та DELETE, що призначені для модифікації даних в таблицях;

– інший підзапит.

Фраза WHERE включає набір умов для відбору рядків:

умова_where ::= [NOT] умова [[AND|OR][NOT] умова]...
умова ::= вираз { = | <> | < | <= | > | >= } { вираз | (підзапит) }
вираз _1 [NOT] BETWEEN вираз _2 AND вираз _3
вираз [NOT] IN { (константа [,константа]...) | (підзапит) }
вираз IS [NOT] NULL
[таблиця.]стовпець [NOT] LIKE 'рядок_символів' [ESCAPE 'символ']
EXISTS (підзапит)

Крім традиційних операторів порівняння (= | <> | <| <= |> |> =) в реченні WHERE використовуються умови BETWEEN (між), LIKE (схоже на), IN (належить), IS NULL (не визначене) і EXISTS (перевірка на існування даних, які відповідають критеріям підзапиту; якщо даних немає, то результат – брехня), які можуть передувати оператором NOT (не). Критерій відбору рядків формується з одного або декількох умов, з'єднаних логічними операторами AND, OR, NOT (з урахуванням їх пріоритету та дужок).

При обробці умови числа порівнюються алгебраїчно – негативні числа вважаються меншими, ніж позитивні, незалежно від їх абсолютної величини. Рядки символів порівнюються відповідно до їх поданням до коді, що використовується в конкретній СУБД, наприклад, в коді ASCII. Якщо порівнюються два рядки символів, що мають різні довжини, більш короткий рядок доповнюється справа пробілами для того, щоб вони мали однакову довжину перед здійсненням порівняння.

Синтаксис модифікатора GROUP BY має вигляд:

GROUP BY [таблиця.]стовпець [,[таблиця.]стовпець] ... [HAVING умова_having]

GROUP BY ініціює перекомпонування формованої таблиці по групах, кожна з яких має однакове значення в стовпцях, включених до переліку GROUP BY. Далі до цих груп застосовуються агрегуючі функції, зазначені в списку вибору SELECT, що призводить до заміни всіх значень групи на єдине значення (сума, кількість і т.п.). В результаті виконання GROUP BY залишаються тільки унікальні значення стовпців, за замовчуванням відсортовані по зростанню.

За допомогою фрази HAVING, де умова_having визначається так само, як і умова_where, можна виключити з результату групи, що не відповідають заданим умовам.

Список параметрів модифікатора ORDER BY має вигляд:

список_order_by ::=
{[таблиця.]стовпець | номер_елементу_SELECT} [[ASC] | DESC]
[, {[таблиця.] стовпець | номер_елементу_SELECT} [[ASC] | DESC]] ...

На відміну від GROUP BY модифікатор ORDER BY не видаляє повторювані значення.

Список параметрів фрази COMPUTE складається з агрегучих функцій, за допомогою яких генеруються підсумкові значення, які відображаються як додаткові рядки, і має наступний синтаксис:

SQL_функція [, SQL_функція] ... [BY стовпець [,стовпець]...]

3.1.3 Вибірка без використання фрази WHERE

Проста вибірка даних

По запиту

```
SELECT    “Назва”, “Адреса”
FROM      “Постачальники”
```

будуть видані назва і адреса постачальників з таблиці Постачальники.
При необхідності отримання повної інформації про постачальників запит буде таким:

```
SELECT    “Код_постачальника”, “Назва”, “Місто”, “Адреса”, “Телефон”
FROM      “Постачальники”
```

або можна замінити список стовпців символом «*», який служить коротким позначенням всіх імен полів в таблиці, зазначеної у фразі FROM, причому порядок виведення полів відповідає порядку, в якому ці поля визначалися при створенні таблиці.

Для виключення дублікатів та одночасного впорядкування переліку необхідно доповнити запит ключовим словом DISTINCT (різний, різні). Якщо після DISTINCT вказується декілька стовпців, то SELECT видасть рядки, унікальні для зазначеного набору стовпців: Наприклад, за запитом:

```
SELECT DISTINCT “Автор”, “Назва”
FROM      “ Книги”
```

будуть видані всі унікальні значення пари автор-назва з таблиці Книги.

Вибірка з використанням виразів

З синтаксису фрази SELECT видно, що в ній може міститися не тільки перелік стовпців таблиці або символ «*», а й вираження. Слід мати на увазі, що обчислювані стовпці не зберігаються в базі даних, а формуються в тимчасовій таблиці, створюваної при виконанні інструкції SELECT. Наприклад, щоб видати список всіх книг, їх цін, розмір 10% надбавки до цін і нові ціни, треба ввести запит:

```
SELECT    “Код_книги”, “Ціна” , Ціна*0.1, Ціна*1.1
FROM      “Книги”
```

Як видно з наведених прикладів, при виведенні результату запиту кожен стовпець виводиться з заголовком, що збігається з його ім'ям в таблиці, а обчислювані стовпці – з заголовком (No column name). Ці заголовки можна змінювати за допомогою псевдонімів. Наприклад, в результаті виконання запиту:

```
SELECT    “Код_книги”, “Ціна” AS “Стара ціна”, “Ціна”*0.1 AS “Надбавка”,
          “Ціна”*1.1 AS "Нова ціна", “Залишок” AS "Є в наявності"
FROM      “Книги”
```

будуть виведені значення стовпця Ціна під псевдонімом Стара_ціна, стовпця Залишок під псевдонімом "Є в наявності" і значення двох обчислюваних стовпців під псевдонімами Надбавка і Нова ціна.

Вирази можуть складатися з окремих числових або текстових констант, причому останні повинні укладатись в одинарні лапки. Текстові константи зазвичай використовуються у вигляді пояснень до даних, які видає в результаті виконання запиту, наприклад:

```
SELECT DISTINCT 'Книга', "Код_книги", 'Ціна', "Ціна", 'uah'
FROM      "Книги"
```

Вибірка NULL-значень

Якщо в якихось із рядків таблиці відсутнє значення для будь-якого стовпчика і для цього стовпця допустимі NULL-значення, то СУБД введе в таке поле NULL-значення. NULL-значення використовується для того, щоб представити єдиним чином "невідомі значення" для будь-яких типів даних. При виведенні NULL-значення на екран або принтер його код відтворюється будь-яким спеціально заданим символом або набором символів, наприклад, <NULL> або поєднанням -0-.

Як правило, в сучасних СУБД за допомогою спеціальної команди можна встановити один з режимів представлення NULL-значень при виконанні числових розрахунків: заборона або дозвіл заміни NULL-значення нулем. У першому випадку будь-арифметичний вираз, що містить невизначений операнд, буде також мати невизначене значення (зазвичай встановлюється за замовчуванням). У другому випадку результат обчислень буде мати чисельне значення (якщо це значення потрапляє в діапазон представлення відповідного типу даних). Для перевірки поточних установок СУБД введіть в таблицю Книги рядок з нульовим значенням стовпця Ціна і потім – запит:

```
SELECT    “Залишок” + 100
FROM      “Книги”
```

Оскільки в SQL всі NULL-значення розглядаються зазвичай як рівні (предикат NULL = NULL має значення «істина»), два невизначених значення розглядаються як дублікати один одного при виключенні дублікатів і тому фраза SELECT DISTINCT дасть в результаті не більше одного NULL-значення. Введіть в таблицю Книги ще один рядок з нульовим значенням стовпця Ціна і зверніть увагу, скільки різних значень будуть виведені при виконанні запиту:

```
SELECT DISTINCT  “Ціна”
FROM      “Книги”
```

3.1.4 Вибірка з використанням фрази WHERE

Припустимо, нас цікавить OID бази даних BookShop. Його можна отримати з системної таблиці pg_database за запитом, в якому зазначено ім'я цікавить нас бази даних:

```
SELECT    OID, datname
FROM      pg_database
WHERE     datname = 'BookShop'
```

Для відбору потрібних рядків таблиці в реченні WHERE можна використовувати оператори порівняння, які можуть передувати оператору NOT, створюючи, наприклад, відносини «не менше" і "не більше", а також можливе використання декількох умов, з'єднаних логічними операторами AND, OR, AND NOT та OR NOT, що дозволяє здійснити більш детальний відбір рядків. Наприклад, щоб отримати список книг з бази даних BookShop, поставлених за останній місяць, вводимо запит:

```
SELECT    "Код_книги", "Кількість", "Дата"
FROM      "Поставки"
WHERE     ("Дата" >= '1/9/2011') AND ("Дата" < '1/10/2011')
```

Якщо в базі даних немає рядків, що задовольняють заданій (им) умові (ам), на екран нічого не буде виводитися. Слід мати на увазі, що в SQL ситуація, коли запит нічого не повертається, не є помилковою, тому контроль за коректністю запиту і результатів його виконання повинен здійснювати користувач.

Використання BETWEEN

За допомогою BETWEEN ... AND ... (знаходиться в інтервалі від ... до ...) можна обрати рядки, в яких значення будь-якого стовпця знаходяться в заданому діапазоні. Наприклад, попередній запит можна переписати так:

```
SELECT    "Код_книги", "Кількість", "Дата"
FROM      "Поставки"
WHERE     "Дата" BETWEEN '1/9/2016' AND '1/10/2016'
```

Можна задати і NOT BETWEEN, якщо необхідно вказати на дані, які не належать деякому діапазону. Оператор BETWEEN можна замінити предикатом, але він покращує сприйняття і читабельність текстів запитів SQL.

Використання IN

В запиті

```
SELECT    *
FROM      “Постачальники”
WHERE     “Місто” IN (‘Київ’, ‘С-Петербург’, ‘Петербург’)
```

оператор IN є насправді просто коротким записом послідовності окремих порівнянь, з'єднаних операторами OR. Попередня інструкція еквівалентна, очевидно, такій:

```
SELECT    *
FROM      “ Постачальники ”
WHERE     “Місто”=‘Київ’ OR “Місто” =‘С-Петербург' OR “Місто”=‘С-Петербург'
```

Можна також вводити NOT IN (не належить) і використовувати замість списку IN (NOT IN) підзапит.

Використання LIKE

Звичайна форма стовпець LIKE текстова_константа для стовпчика текстового типу дозволяє відшукати всі значення зазначеного стовпця, відповідно зразку, заданому текстовою_константою. Символи цієї константи інтерпретуються наступним чином:

- символ «_» замінює будь-який одиночний символ,
- символ «%» замінює будь-яку послідовність з 0 або більше символів,
- всі інші символи означають просто самі себе.

Наприклад, щоб отримати список книг, в назві яких є підрядок «С ++», треба сформувати запит:

```
SELECT    “Код_книги”, “Назва”, “Автор”
FROM      “Книги”
WHERE     “Назва” LIKE ‘%С ++%"
```

Вибірка NULL-значень

Якщо при завантаженні даних не введено значення в будь-яке поле таблиці, то СУБД помістить в нього NULL-значення. Виявити ці значення можна за допомогою умов:

стовпець IS NULL або стовпець IS NOT NULL

Наприклад, запит, наведений нижче, поверне рядки з таблиці Постачальники, в яких відсутні відомості в зазначених колонках:

```
SELECT    *
FROM      “Постачальники”
WHERE     “Адреса” IS NULL OR “Телефон” IS NULL
```

Обмеження максимального розміру вибірки

У PostgreSQL є специфічні конструкції LIMIT і OFFSET, за допомогою яких можна управляти кількістю і порядком виведення записів. LIMIT обмежує максимальну кількість записів, що будуть повернені, а OFFSET задає номер запису в курсорі (0,1,2, ...), починаючи з якої будуть виводитися записи. Вони мають такий синтаксис:

```
LIMIT { число | ALL } [OFFSET початок]
```

де число – кількість записів, ALL – всі записи, початок – номер першого запису. Наприклад, якщо число = 10 і початок = 3, то запит поверне 10 записів, починаючи з четвертої (номер 3) по порядку.

Використання даних типу timestamp

Дані типу timestamp використовуються в реченні WHERE для запобігання одночасного поновлення одного і того ж рядка двома користувачами. Якщо при зміні значень цього рядка поле типу timestamp також змінюється, то при одночасному редагуванні рядка двома користувачами при збереженні одним з них оновленої інформації поле timestamp зміниться і рядок не буде відповідати колишній копії. Коли інший користувач спробуєте зберегти свої зміни, пропозиція WHERE не дасть йому цього зробити, оскільки команда не знайде обраний раніше рядок. Дані типу timestamp – це хороший засіб для підтримки сервером унікальності кожного рядка.

3.1.5 Вибірка з впорядкуванням

Найпростіший варіант використання модифікатора ORDER – впорядкування рядків результату за значенням одного з стовпців із зазначенням порядку сортування або без такої вказівки (за замовчуванням рядки будуть сортуватися в порядку зростання значень у вказаному стовпці.). Наприклад, видати перелік книг з таблиці Книги в алфавітному порядку за прізвищами авторів:

```
SELECT    “Код_книги”, “Автор”, “Назва”
FROM   “Книги”
ORDER BY “Автор”
```

або в порядку зменшення ціни:

```
SELECT    “Код_книги”, “Ціна”
FROM   “Книги”
ORDER BY “Ціна” DESC
```

При включенні в список ORDER BY декількох стовпців СУБД сортує рядки результату за значеннями першого стовпчика списку поки не з'явиться кілька рядків з однаковими значеннями даних в цьому стовпці. Такі рядки упорядковуються відповідно до значень у наступній колонці зі списку ORDER BY і т.д. Наприклад, видати вміст таблиці Замовлення, відсортувавши її рядки по стовпчиках Код_замовника і Дата:

```
SELECT    *
FROM   “Замовлення”
ORDER BY “Код_замовника”, “Дата”
```

Крім того, в список ORDER BY можна включати не тільки ім'я стовпчика, а й його порядкову позицію в переліку SELECT. Номер стовпця в списку елементів SELECT

відраховується зліва направо. Завдяки цьому можливо впорядкування результатів на основі обчислюваних стовпців, що не мають імен, наприклад:

```
SELECT     “Код_книги”, ”Ціна” * 0.2
FROM   “Книги”
ORDER BY 2 DESC
```

У списку вибору інструкції SELECT стовпці можна задавати різними способами, не враховуючи порядок їх слідування в базовій таблиці. Можна навіть один і той же стовпець вказувати кілька разів, наприклад:

```
SELECT     “Код_книги”, ”Автор”, “ Назва”, “Видавництво”, “Ціна”, “Код_книги”
FROM       “Книги”
ORDER BY 2, 3
```

Видані на цей запит рядки не поміщаються повністю на екрані, і тому при горизонтальній прокрутці перші стовпчики виходять за межі активного вікна. Виведений повторно в кінці списку Код_книги полегшить роботу з таким списком.

3.1.6 Вибірка та агрегування даних

Для виконання об'єднання даних за деяким критерієм використовується модифікатор GROUP BY, який ділить таблицю на групи рядків, кожна з яких має однакові значення в стовпці, зазначеному в GROUP BY.

Нехай необхідно дізнатися, які постачальники здійснювали поставки. Для цього можна, очевидно, ввести запит:

```
SELECT DISTINCT  "Код_постачальника"
FROM       "Поставки"
```

Теж саме отримаємо за допомогою запиту:

```
SELECT     "Код_ постачальника "
FROM       "Поставки"
GROUP BY "Код_ постачальника "
```

Однак, якщо тепер в список SELECT додати ще одне поле, наприклад, Код_книги, то отримаємо повідомлення про помилку:

ERROR: column "Поставки.Код_книги" must appear in the GROUP BY clause or be used in an aggregate function

тобто поле Код_книги має бути зазначено в рядку GROUP BY або використовуватися в агрегуючій функції. Це пов'язано з тим, що SQL-функція створює єдине значення з множини значень стовпчика-аргументу, а для «вільного» стовпця має бути видано всі множини його значень. Тому подібний запит відкидається системою.

Якщо додати Код_книги в рядок GROUP BY, то в групах з однаковим значенням Код_постачальника дані додатково будуть згруповані по Код_книги.

Сама по собі угруповання не має сенсу – угруповання (або агрегування) зазвичай передує якійсб обробці, яка виконується над даними в групах. Такі операції вводяться за допомогою агрегуючої SQL-функції: SUM – сума, AVG – середнє, MIN – мінімальне значення, MAX – максимальне значення, COUNT – кількість.

Повертаючись до останнього прикладу, припустимо, що необхідно отримати дані про кількість книг, поставлених кожним постачальником окремо. Це можна зробити за допомогою запиту:

```
SELECT     "Код_постачальника"', “Код_книги”, SUM(“Кількість”) AS “Кількість”
FROM       "Постачальники"
GROUP BY "Код_постачальника ", “Код_книги”
```

Якщо GROUP BY не використовується, то при наявності в списку SELECT агрегуючих функцій в цей список можна включати лише SQL-функції або вирази, що містять такі функції.

У стовпці-аргументі перед застосуванням будь-якої функції, крім COUNT(*), виключаються всі невизначені значення. Якщо виявляється, що аргумент – порожня множина, функція COUNT приймає значення 0, а решта – NULL.

Припустимо, що тепер потрібно дізнатися, скільки книг кожного видавництва представлено в базі даних BOOKSHOP. Для цього введемо запит:

```
SELECT      “Видавництво”, COUNT(*)
FROM        “Книги”
GROUP BY    “Видавництво”
```

Рядки з однаковими значеннями стовпця Видавництво об'єднуються спочатку в групи і для кожної групи виводиться тільки один рядок, у другому стовпці якої виводиться результат виконання функції COUNT для даної групи, тобто кількість рядків в ній.

Тобто, GROUP BY ініціює перекомпонування зазначеної таблиці по групах, далі до кожної групи застосовується інструкція SELECT. Кожен вираз в списку SELECT має приймати єдине значення для групи, тобто воно може бути або значенням стовпця, зазначеного в GROUP BY, або арифметичним виразом, що включає це значення, або константою, або однією з скалярних SQL-функцій, яка оперує всіма значеннями стовпчика в групі і зводить ці значення до єдиного значенням (наприклад, до суми, середнього значення і т.д.).

Модифікатор GROUP BY не припускав упорядкування результату. Щоб упорядкувати результати запиту, в кінці інструкції треба додати модифікатор ORDER BY:

```
SELECT      "Код_постачальника"', “Код_книги”, SUM(“Кількість”) AS “Кількість”
FROM        "Поставки"
GROUP BY "Код_постачальника ", “Код_книги”
ORDER BY 1, 3
```

Рядки таблиці можна групувати по будь-якій комбінації її стовпців. Так, в процесі виконання запиту:

```
SELECT      “Код_замовника”, “Сплачено”, COUNT(*) AS “Кількість замовлень”
FROM        “Замовлень”
GROUP BY    “Код_замовника”, “Сплачено”
ORDER BY    1
```

спочатку виконується групування за значеннями стовпчика Код_замовника, потім в кожній з отриманих підгруп виконується групування за значеннями стовпчика Сплачено і застосовується агрегуюча функція COUNT, тобто обчислюється для кожного замовника кількість оплачених і неоплачених замовлень.

Якщо в запиті використовуються WHERE і GROUP BY, то рядки, що не задовольняють умові WHERE, виключаються до виконання групування. Наприклад, попередній запит можна модифікувати так, щоб він видавав список замовників, які оплатили замовлення, і кількість оплачених ними замовлень:

```
SELECT      “Код_замовника”, “Сплачено”, COUNT(*) “Кількість замовлень”
FROM        “Замовлення”
WHERE       “Сплачено” = ‘Так’
GROUP BY    “Код_закмовника”, “Сплачено”
ORDER BY    1
```

Фраза HAVING грає таку ж роль для груп, що і WHERE для рядків: вона використовується для виключення груп, так само, як WHERE використовується для виключення рядків. Ця фраза включається в інструкцію лише при наявності фрази GROUP BY, а вираз в HAVING повинен приймати єдине значення для групи. Наприклад, надати список постачальників, які поставили більше двох книг:

```
SELECT    “Код_постачальника”, COUNT(*) “Кількість поставлених книг”
FROM      “Книги”
GROUP BY  “Код_постачальника”
HAVING    COUNT(*) > 2
```

3.1.7 Перетворення типів даних в інструкції SELECT

У PostgreSQL існує чотири фундаментальні SQL-конструкції, що вимагають чітких правил перетворення типів: виклики функцій, застосування операторів, присвоювання значень при вставці і модифікації даних, застосування конструкцій UNION (об'єднання результатів запитів) і CASE (аналог оператора if..then..else) .

У PostgreSQL підтримуються три варіанти синтаксису явного перетворення (приведення) типів:

- для строкових констант
  тип 'значення'
  'значення' :: тип
  CAST ('значення' AS тип)
- для числових констант
  значення :: тип
  CAST (значення AS тип)
- для полей набору даних, що повертаються запитом SQL
  ідентифікатор :: тип
  CAST (ідентифікатор AS тип)

Приклад:

```
SELECT ("Залишок"+10)::float * 1.1,
          "Ціна"::char(8) || char(4) ' UAH', -- конкатенація двух строк
          CAST("Ціна" AS text)
FROM "Книги"
```

Наступний приклад демонструє використання функцій перетворення типів для побудови строкового виразу:

```
SELECT 'Від ' || '01/01/2016' || ' до ' || CAST(now() AS varchar(64)) || ' пройшло ' || (CAST (
            now() AS varchar(64)) :: date - date '01/01/2016') :: text || ' днів'
```

Можливо також використовувати приведення типів в фразі WHERE. Наступний запит поверне дані про січневі поставки в базі даних BookShop:

```
SELECT *
FROM   "Поставки"
WHERE CAST("Дата" AS varchar) LIKE '%2016-01-%'
```

Той же результат виходить за допомогою стандартної функції extract:

```
SELECT *
FROM   "Поставки"
WHERE  extract(month from "Дата") = 1 AND extract(year  from "Дата") = 2016
```

Крім синтаксичних форм перетворення типів існують деякі функції, що дозволяють добитися практично того ж результату. Імена цих функцій часто збігаються з іменами підсумкових типів, наприклад, text () або timeofday ().

3.1.8 Використання SELECT для з'єднання двох та більше таблиць

Декартовий добуток таблиць

З'єднання таблиць – це підмножини їх декартового добутку. Так як декартовій добуток n таблиць – це таблиця, яка містить всі можливі рядки r, такі, що r є зчепленням будь-якого рядка з першої таблиці, рядка з другої таблиці, ..., рядка з n-й таблиці (за допомогою SELECT можна отримати будь-яку підмножина реляційної таблиці), то залишилося лише з'ясувати, чи можна за допомогою SELECT отримати декартовий добуток. Для отримання декартового добутку кількох таблиць треба вказати у фразі FROM перелік перемножуваних таблиць, а у фразі SELECT – всі їх стовпці:

```
SELECT    ”Замовлення”.*, “Замовники”.*
FROM      “ Замовлення ”,  “ Замовники ”
```

В результаті SQL видасть комбінацію кожного рядка з таблиці Замовлення з усіма рядками таблиці Замовники, тобто кількість рядків у запиті дорівнює добутку кількості рядків обох таблиці. Більшість цих рядків не мають ніякого сенсу.

Еквіз'єднання таблиць

Якщо з декартового добутка прибрати непотрібні рядки і стовпці, то можна отримати цілком осмислений результат, що відповідає будь-якому із з'єднань. Наприклад, якщо до попереднього запиту додати фразу WHERE:

```
SELECT    “Замовлення”.*, “Замовники”.*
FROM      “ Замовлення ”, “ Замовники ”
WHERE     “ Замовлення ”.”Код_замовника” = “ Замовники ”.”Код_замовника”
```

то отримаємо таблицю – еквівалентні з'єднання двох таблиць, в якій залишаться тільки рядки з однаковими значеннями в стовпчиках Код_замовника, але таких стовпців буде два і вони будуть дублювати один одного.

Природне з'єднання таблиць

Якщо з еквіз'єднання таблиць виключити один з стовпців-дублікатів, за якими проводилося з'єднання, то отримаємо природне з'єднання тих же таблиць. Це можна зробити, змінивши попередній запит наступним чином:

```
SELECT    “Код_замовлення”, “Код_книги”, “ Дата”, “Сплачено”, “Замовники”.*
FROM      “Замовлення”, “Замовники”
WHERE     “Замовлення”.”Код_замовника” = “Замовники”.”Код_замовника”
```

Композиція таблиць

Якщо виключити всі стовпці, по яким проводиться з'єднання таблиць, то отримаємо їх **композицію:**

```
SELECT    “Код_замовлення”, “Код_книги”, “Дата”, “Сплачено”, “Ім'я”, “Адреса”,
“Телефон”
FROM      “Замовлення”, “Замовники”
WHERE     “Замовлення”.”Код_замовника” = “Замовники”.”Код_замовника”
```

При формуванні з'єднання створюється робоча таблиця, до якої можуть бути застосовані всі операції, розглянуті раніше: відбір потрібних рядків з'єднання (WHERE), впорядкування одержуваного результату (ORDER BY) і агрегування даних (SQL-функції і GROUP BY).

Наприклад, наступний запит дозволить зробити вибірку всіх оплачених замовлень за період часу між двома датами, упорядкованих по Код_книги:

```
SELECT   “Код_замовника”, “Код_книги”, “Дата”, “Сплачено”, “Код_замовника”
FROM     “Замовлення”, “Замовники”
WHERE    (“Заказы”.”Код_замовника” = “Замовники”.”Код_замовника”)
         AND (“Дата” BETWEEN  ‘03/01/2016’ AND ‘04/30/2016’)
         AND (“Сплачено” LIKE  ‘%так%’)
ORDER BY “Код_книги”
```

З’єднання таблиці зі своєю копією

У ряді програм виникає необхідність одночасної обробки даних будь-якої таблиці і однієї або декількох її копій, створюваних на час виконання запиту. Наприклад, при додаванні запису в таблицю Замовники можливий повторне введення даних про будь-якому замовнику з присвоєнням йому іншого коду. Для виявлення таких помилок можна з'єднати таблицю Замовники з її тимчасової копією, встановивши в реченні WHERE рівність значень всіх однойменних стовпців цих таблиць, крім стовпців з кодом замовника (для останніх треба встановити умову нерівності значень). Тимчасові копії таблиці можна сформувати за допомогою псевдонімів, вказавши їх за ім'ям таблиці в реченні FROM. Так, якщо ввести:

```
FROM     “Замовники” AS A, “ Замовники” AS “B”
```

або просто

```
FROM     “Замовники” A, “ Замовники ” B
```

то будуть сформовані дві копії таблиці Замовники з іменами A, B. Тоді, щоб дізнатися, чи є в базі даних два різних замовника з однаковими телефонами, треба ввести запит:

```
SELECT   A.*, B.*
FROM     “Замовники” A, “ Замовники ” B
WHERE    A.”Телефон” = B.”Телефон”
         AND A.”Код_замовника” != B.”Код_замовника”
```

3.1.9 Вкладені підзапити

Вкладений підзапит – це підзапит, взятий у круглі дужки і вкладений в фразу WHERE (HAVING) інструкції SELECT або інших інструкцій, які використовують фразу WHERE. Вкладений підзапит може містити в своїй фразі WHERE (HAVING) інший вкладений підзапит і т.д. **Призначення вкладених підзапитів** полягає в тому, щоб при відборі рядків таблиці, сформованої основним запитом, можна було використовувати дані з інших.

Існують **прості** і **корельовані вкладені підзапити.**

Прості вкладені підзапити обробляються системою "від низу до верху". Першим обробляється вкладений підзапит самого нижнього рівня. Множина значень, отримана в результаті його виконання, використовується при реалізації підзапиту вищого рівня і т.д.

Корельовані вкладені підзапити обробляються системою в зворотному порядку. Спочатку вибирається перший рядок робочої таблиці, сформованої основним запитом, і з неї вибираються значення тих стовпців, які використовуються у вкладеному підзапит (вкладених підзапитах). Якщо ці значення задовольняють умовам вкладеного підзапиту, то обраний рядок включається в результат. Потім вибирається другий рядок і т.д., поки в результат не будуть включені всі рядки, що задовольняють вкладеному підзапит (послідовності вкладених підзапитів).

– Підзапити включаються в фразу WHERE (HAVING) за допомогою ключових слів IN, EXISTS або однієї з умов порівняння (= | <> | <| <= |> |> =):

– підзапити, що не повертають жодного або повертають кілька значень, починаються з IN або операторів порівняння і містять ключові слова ANY або ALL;

– підзапити, що повертають єдине значення, починаються з оператора порівняння;

– підзапити, що представляють собою тест на існування або присутність даних, починаються з EXISTS.

Прості вкладені підзапити

Прості вкладені підзапити використовуються для представлення множині значень, дослідження яких має здійснюватися в будь-якому предикаті IN, наприклад:

```
SELECT     *
FROM       “Постачальники”
WHERE      “Код_ постачальника” IN
             (SELECT DISTINCT       “Код_постачальника”
               FROM “Книги”
               WHERE“Автор” LIKE ‘%Дейт%’ )
```

Система виконує насамперед вкладений підзапит. Цей підзапит видає множину унікальних кодів постачальників, які поставили книги зазначеного автора. Потім буде виконаний зовнішній запит, який виведе дані про ці постачальників.

Той же результат можна отримати за допомогою з'єднання:

```
SELECT     ”Постачальники”.*
FROM       “Книги”, “Постачальники”
WHERE      “Книги”.”Код_постачальника” = “Постачальники”.“Код_постачальника”
           AND “Автор” LIKE ‘%Дейт%’
```

При виконанні останнього запиту система повинна одночасно обробляти дані з двох таблиць, тоді як в попередньому прикладі ці таблиці обробляються по черзі. Природно, що для їх реалізації потрібні різні ресурси пам'яті і часу.

У подібних запитах можна використовувати не тільки оператор IN, а й оператори порівняння (<>, <=, <,> = або >), проте, **якщо вкладений підзапит повертає більше одного значення і не використовується оператор IN, буде виникати помилка.**

Коррельовані вкладені підзапити

Розглянемо запит:

```
SELECT     *
FROM       “Замовлення”
WHERE      ‘Хтось’ IN
           (SELECT ”Ім’я”
             FROM  “Замовники”
             WHERE “Код_замовника” = “Замовлення”.”Код_замовника”);
```

Такий підзапит відрізняється тим, що вкладений підзапит не може бути оброблений перш, ніж буде оброблятися зовнішній підзапит, оскільки вкладений підзапит залежить від значення Замовники.Код_замовника, а воно змінюється в міру того, як система перевіряє різні рядки таблиці Замовлення. Обробка цього підзапиту здійснюється наступним чином:

1) Система перевіряє перший рядок таблиці Замовлення. Припустимо, що в цьому рядку Код_замовника має значення 1. Тоді значення Замовлення.Код_замовника в даний момент також набуде значення 1 і система зможе обробити внутрішній запит:

```
(SELECT    Ім’я
FROM       Замовники
WHERE      Код_замовника = 1)
```

отримавши в результаті деяка підмножина значень поля Ім'я. Вибірка значень для цього замовника буде проведена тоді і тільки тоді, коли значення 'Хтось' буде належати цій множині. На цьому завершується опрацювання для замовника з кодом 1.

2) Далі вибирається другий рядок таблиці Замовлення і виконується аналогічна обробка для наступного замовника і т.д. до тих пір, поки не будуть розглянуті всі рядки таблиці Замовлення.

Подібні підзапити називаються корельованими, так як їх результат залежить від значень, визначених в зовнішньому підзапиті. Обробка корельованого підзапиту, отже, повинна повторюватися послідовно для кожного значення витягується з зовнішнього підзапиту.

Наступний приклад демонструє використання однієї і тієї ж таблиці в зовнішньому підзапиті і корелювати вкладеному підзапит:

```
SELECT    "Код_книги", "Автор", "Назва", "Видавництво"
FROM      "Книги" A
WHERE     "Назва" IN
          (SELECT DISTINCT "Назва"
           FROM  "Книги" B
           WHERE A."Видавництво" != B." Видавництво " )
ORDER BY "Автор"
```

У цьому запиті по черзі для кожного рядка таблиці Книги вибирається значення поля Видавництво і для цього значення виконується вкладений підзапит, результатом якого є список унікальних назв книг, виданих іншими видавництвами. Поточний рядок таблиці Книги включається в результат запиту, якщо значення в поле Назва входить в список, сформований вкладеним запитом. Т.ч., будуть видані дані про книги, видані більш ніж в одному видавництві, і результуючий список буде відсортований по полю Автор.

Запити, що використовують EXISTS в фразі WHERE

У мові SQL підзапити, що виконують перевірку на існування, представляються виразом:

WHERE [NOT] EXISTS (підзапит)

Такий вираз вважається дійсним тільки тоді, коли підзапит повертає непорожню множину, тобто коли існує будь-який рядок в таблиці, зазначеної в пропозиції FROM підзапиту, яка задовольняє умові WHERE підзапиту (практично цей підзапит завжди буде корельованим множиною).

Розглянемо приклад запиту на видачу назви постачальників, що поставляють книги деякого видавництва, скажімо BHV:

```
SELECT    "Назва"
FROM      "Постачальники" A
WHERE     EXISTS (
          SELECT *
          FROM  "Книги"
          WHERE "Код_постачальника" = A."Код_постачальника"
          AND "Видавництво" = 'BHV')
```

Система послідовно вибирає рядки таблиці Постачальники, виділяє з них значення стовпця Назва, а потім перевіряє, чи існує в таблиці Книги хоча б один рядок зі значенням 'BHV' в стовпці Видавництво і значенням в стовпці Код_постачальника, рівне значенню того ж стовпчика, обраного з таблиці постачальники. Якщо умова виконується, то отримане значення стовпця Назва включається в результат.

Підзапити з ANY та ALL

ANY і ALL відрізняються від EXISTS тим, що використовуються спільно з реляційними операторами. В цьому сенсі вони подібні оператору IN, проте, на відміну від IN, вони можуть використовуватися тільки з підзапитах.

Оператор ANY призводить до того, що поточний рядок зовнішнього запиту включається в результат, якщо будь-яке (ANY) зі значень, виведених підзапитом, таке саме, як в поточному рядку зовнішнього запиту. Наступний запит:

```
SELECT    *
FROM      ”Замовники”
WHERE     “Код_замовника”  = ANY (SELECT “Код_замовника”  FROM “Замовлення”)
```

рівнозначний запиту

```
SELECT    *
FROM      Замовники
WHERE     Код_замовника IN (SELECT Код_замовника FROM Замовлення);
```

Відмінність полягає в тому, що оператор ANY може використовувати інші оператори порівняння, а не тільки оператор «=» як в виразі IN. Наприклад, якщо в попередньому запиті в реченні WHERE замінити оператор «=» на оператор «<=», то отримаємо інформацію про замовника з мінімальним значенням в полі Код_замовника, оскільки це – єдине значення, для якого умова основного запиту виконується.

Оператор ALL аналогічний оператору ANY по взаємодії із зовнішнім запитом, але для того, щоб поточне значення в зовнішньому запиті було вибрано, воно повинно задовольняти умові в предикаті зовнішнього запиту для кожного значення, обраного підзапитом.

У SQL сказати, що значення більше (або менше), ніж будь-який (ANY) з набору значень, – теж саме що сказати, що воно більше (або менше) ніж будь-яке одне окреме з цих значень. І навпаки, сказати, що значення не дорівнює всьому (ALL) набору значень, рівнозначно тому, що немає такого значення в наборі, якому воно б відповідало.

Будь-який запит, який може бути сформульований з ANY або з ALL, може бути також сформульовано з EXISTS, хоча зворотне в загальному випадку невірно. Строго кажучи, варіант з EXISTS не абсолютно ідентичний варіантів з ANY або з ALL через відмінності в тому, як обробляються NULL-значення. Всякий раз, коли допустимий підзапит повертає порожня множина, предикат з ALL автоматично вірний, а з ANY - автоматично невірний.

Використання WITH

Конструкція WITH дозволяє оформити результати підзапиту у вигляді тимчасової таблиці, яка створюється на час виконання запиту. Т.ч., WITH може використовуватися для структурування складних запитів з багаторівневою ієрархією підзапитів. Синтаксис конструкції WITH досить простий:

```
WITH  ідентифікатор  AS (запит)
```

Наприклад:

```
WITH  subquery_ as (
    SELECT "Назва", "Ціна"
    FROM    "Книги"
    WHERE "Видавництво" = 'BHV')
SELECT *
FROM _subquery_
WHERE   "Ціна" >= ANY(SELECT "Ціна" FROM _subquery_ )
```

Об'єднання двох та більше запитів

Інструкція UNION дозволяє об'єднати результати виконання декількох запитів до різних таблиць в одній підмножині даних, реалізуючи один запит. Синтаксис UNION має вигляд:

запит_1 UNION запит_2

Неодмінна вимога – однакова кількість і типи полів в об'єднаних інструкцією UNION запитах, хоча другу вимогу можна обійти, якщо використовувати перетворення типів в запитах. наприклад:

```
SELECT ”Ім’я”, “Адреса”
FROM “Замовники”
UNION
SELECT “Назва”, “Адреса”
FROM “Постачальники”
```

Варто мати на увазі, що:

– Надлишкові дублікати завжди виключаються з результату UNION.

– Інструкцією з UNION можна об'єднати будь-яке число таблиць (проекцій таблиць).

– Весь запит може використовувати тільки одну фразу ORDER BY. Вона застосовується до всього результату і повинна знаходитися після останньої інструкції SELECT.

3.1.10 Реалізація операцій реляційної алгебри за допомогою фрази SELECT

За допомогою інструкції SELECT можна реалізувати будь-яку операцію реляційної алгебри:

– **Селекція (горизонтальна підмножина) таблиці** створюється з тих же рядків, які задовольняють заданим умовам, наприклад:

```
SELECT   *
FROM     ”Книги”
WHERE    “Видавництво” = 'BHV' AND “Залишок” > 0
```

– **Проекція (вертикальна підмножина) таблиці** створюється з зазначених її стовпців (в заданому порядку) з наступним виключенням надлишкових дублікатів рядків, наприклад:

```
SELECT    DISTINCT “Автор”, “Назва”,” Видавництво”, “Ціна”
FROM      “Книги”
```

– **Об'єднання двох таблиць** містить ті рядки, які є або в першій, або в другій, або в обох таблицях, наприклад:

```
SELECT    “Автор”, “Назва”,” Видавництво”, “Ціна”
FROM      “Книги”
WHERE     “Назва” LIKE ‘%SQL%''
UNION
SELECT    “Автор”, “Назва”,” Видавництво”, “Ціна”
FROM      “Книги”
WHERE     “Ціна” BETWEEN 10 AND 100
```

Початкові відношення або їх об'єднувані проекції повинні бути сумісними за об'єднанням. Для SQL це означає, що дві таблиці можна об'єднувати тоді і тільки тоді, коли вони мають однакове число стовпців, наприклад, m і для всіх i (i = 1, 2, ..., m) i-й стовпець першої таблиці і i -й стовпець другої таблиці мають в точності однаковий тип даних.

– **Перетин двох таблиць** містить тільки ті рядки, які є і в першій, і в другій, наприклад:

```
WITH _table_1 AS (
        SELECT “Код_замовника”
        FROM   “Замовники”)
WITH _table_2 AS (
        SELECT “Код_заказчика”
        FROM   “Замовлення”)
SELECT *
FROM _table_1
WHERE “Код_замовника” IN (SELECT * FROM  _table_2)
```

– **Різниця двох таблиць** містить тільки ті рядки, які є в першій, але відсутні в другій, наприклад:

```
WITH _table_1 AS (
        SELECT “Код_замовника”
        FROM   “Замовники”)
WITH _table_2 AS (
        SELECT “Код_замовника”
        FROM   “Замовлення”)
SELECT *
FROM _table_1
WHERE  “Код_замовника”  NOT IN (SELECT * FROM _table_2)
```

3.2 Завдання для самостійної роботи

1) Напишіть запит для виведення назви, автора і ціну книг з таблиці Книги.
2) Напишіть запит для виведення всіх унікальних імен авторів книг в поточному порядку з таблиці Книги.
3) Напишіть запит для виведення всіх замовлень з таблиці Замовлення зі значеннями суми передоплати вище 1,000 гривень.
4) Напишіть запит для виведення всіх замовлень з таблиці Замовлення, для яких сума передоплати становить не менше 30% від вартості книги.
5) Напишіть запит до таблиці Замовлення для виведення всіх оплачених замовлень, відповідних замовнику з ім'ям «Іванов І.».
6) Напишіть запит для виведення всіх рядків з таблиці Замовники, що відносяться до замовників, які проживають в Бресті.
7) Напишіть запит для виведення кодів всіх книг, їх цін і кодів постачальників для книг, поставлених з Києва.
8) Що може бути виведено в результаті наступного запиту?
9) SELECT *
10) FROM "Книги"
11) WHERE ("Ціна" <10000 OR NOT ("Автор" IS NULL AND "Залишок"> 50))
12) Як можна інакше переписати запит:
13) SELECT *
14) FROM "Замовлення"
15) WHERE ("Дата"> '01 / 01/01 'OR "Дата" <'01 / 03/01)
16) Напишіть запит для вибірки всіх замовників, чиї імена починаються з букви, що потрапляє в діапазон від А до М.
17) Напишіть запит для вибірки всіх книг, в назві яких є словосполучення «база даних» або «бази даних» або «баз даних».
18) Напишіть запит для виведення всіх постачальників, дані про яких містять NULL-значення.
19) Напишіть запит для підрахунку суми всіх замовлень на 31 січня минулого року.

20) Напишіть запит для підрахунку числа різних не NULL-значень полів Адреса та Телефон в таблиці Постачальники.

21) Напишіть запит для вибірки найменшої суми передоплати для кожного замовника.

22) Напишіть запит для вибірки замовників в алфавітному порядку, чиї імена починаються з букви М.

23) Напишіть запит для підрахунку загальної суми замовлень за кожен день з 01.02 по 01.04 поточного року.

24) Напишіть запит до таблиці Замовлення для визначення середньої суми передоплати за кожен місяць.

25) Напишіть запит для виведення загальної вартості оплачених замовлень на кожен день в низхідному порядку.

26) Напишіть запит для формування рейтингу авторів у вигляді упорядкованого по спадаючій списку авторів і кількості їх книг, проданих за деякий період.

27) Напишіть запит, який би виводив всі замовлення з величиною передоплати вище середньої. Виведіть код замовлення, дату, ім'я замовника і величину передоплати.

28) Напишіть запит, який вирахував би сумарні залишки кількості книг, поставлених кожним постачальником. Виведіть код постачальника, кількість видів книг і сумарний залишок.

29) Напишіть запит, який би вивів всі замовлення, зроблені за перший квартал поточного року замовниками з Бреста, і підрахував кількість оплачених і неоплачених замовлень.

30) Напишіть запит, який вивів би імена всіх замовників, які проживають в тому ж місті, що і замовник, який має на даний момент часу максимальну суму всіх сплачених замовлень.

31) Напишіть запит для отримання списку книг, ціни яких перевищать 10000 руб. в разі підвищення цін на 12%.

32) Напишіть запит, який би вибрав загальну суму всіх придбань для кожного замовника, у якого ця загальна сума більша за середню по всім замовникам.

33) Напишіть запит з використанням корельованого підзапиту для вибірки імен та кодів всіх замовників, у яких найбільша загальна вартість придбаних книг серед замовників, які проживають в даному місті.

34) Напишіть два запити які виберуть всіх замовників (по їх імені та коду), для яких є постачальники придбаних ними книг, які проживають в тих же містах, що і замовники. Перший запит – з використанням природного з'єднання, а інший – з корельованим підзапитом.

35) Напишіть запит для вибірки даних про книгах, які в даний момент замовлені принаймні одним замовником, а також тих книг, які до цього моменту часу жодного разу не були замовлені.

36) Напишіть запит, який використовує ANY або ALL, який би знаходив всіх замовників, які не сплатили жодного замовлення.

37) Створіть об'єднання з двох запитів, яке вивело б коди і імена всіх замовників. Ті з них, для яких суми придбаних книг вище середньої, повинні, крім того, мати коментар "Високий Рейтинг", а інші – "Низький Рейтинг".

3.3 Контрольне завдання

Написати запити до розробленої в ході виконання лабораторної роботи №2 бази даних (табл. 3.1), використовуючи прості і корельовані підзапити. Результати запитів повинні виводитися зі смисловими назвами стовпців і супроводжуватися при необхідності текстовими коментарями.

Таблиця 3.1 – Варіанти завдань до розробленої бази даних

№	**Назва розроблюваної БД та запити.**
1	Торгівля
	1) Отримати список виробів, які в даний момент часу відсутні на складі. 2) Отримати загальну вартість і кількість проданих за останні три місяці виробів всіх наявних в прайс-листі найменувань. 3) Отримати дані про вироби, проданих за безготівковим розрахунком. Вивести кількість одиниць кожного найменування. 4) Отримати кількість проданих одиниць і загальну суму виторгу для даного виробу по кожному з виробників. Результат впорядкувати по спадаючій сумі виручки. 5) Отримати рахунки, оплачені за готівковим розрахунком з початку року. Список впорядкувати по спадаючій сумарній вартості виробів, які фігурують в рахунку.
2	Комунальні платежі.
	1) Отримати списки клієнтів, які не сплатили на даний момент часу більше 1 послуги. По кожному клієнту видати загальну суму заборгованості. 2) Отримати перелік надаваних комунальних послуг із зазначенням тарифів, середньомісячних обсягів споживання. 3) Отримати списки клієнтів, у яких споживання даного виду комунальних послуг за останній місяць перевищує середнє значення. 4) Отримати відомість оплати комунальних послуг за останній місяць клієнтами, які проживають за вказаною адресою. Вивести підсумкові суми оплати за кожного клієнта і за кожним видом послуг. 5) Отримати дані про заборгованість по кожному виду комунальних послуг за кожен місяць з початку року.
3	Послуги.
	1) По кожному типу робіт отримати його частку в процентному вираженні від загального числа зареєстрованих замовлень. 2) По кожному майстру отримати загальну кількість виконаних замовлень і сумарну вартість виконаних при цьому робіт. Дані впорядкувати по спадаючій сумарній вартості. 3) Отримати список перших 10 клієнтів від 20 до 45 років з максимальною кількістю замовлень. Для кожного з них вивести також сумарну вартість замовлень. 4) Отримати список клієнтів, які зверталися в агентство з початку поточного року, упорядкований по спадаючій кількості різних типів виконаних робіт. 5) Отримати список клієнтів, сумарна вартість замовлень яких перевищує середню вартість замовлення по агентству.
4	Нарахування зарплати.
	1) Отримати відомість виплати зарплати за останній місяць, в якій для кожного працівника вказати: прізвище та ініціали, посаду, нараховану суму, суму утримань, суму до видачі. Дані впорядкувати за посадою і прізвищем співробітників. 2) Для кожної категорії працівників, зазначених у штатному розкладі, отримати сумарну кількість пропущених робочих днів без урахування відпускних, а також середньомісячний розмір виплачених премій. Результат впорядкувати по спадаючій кількості пропущених робочих днів. 3) Для кожної категорії працівників, зазначених у штатному розкладі отримати кількість працівників із середньою, середньою спеціальною та вищою освітою. 4) Отримати упорядкований за прізвищами список працівників, у яких зарплата за останні три місяці перевищувала середньомісячну зарплату по підприємству. 5) Отримати в процентному вираженні частку зарплати працівників із середньою, середньою спеціальною та вищою освітою від загального фонду зарплати.

Продовження таблиці 3.1

№	Назва розроблюваної БД та запити.
5	Поставки.
	1) Отримати список постачальників даного типу продукції, упорядкований в порядку убування оптової ціни цієї продукції. 2) Отримати список договорів, укладених за останні три місяці, з простроченим терміном поставки. Дані вивести в порядку убування затримки поставки. 3) Отримати середньомісячну кількість договорів, укладених за кожним видом продукції. 4) По кожному типу продукції отримати частку договорів, що укладаються на поставку продукції з найвищим показником якості, від загальної кількості договорів на поставку продукції даного типу. 5) Отримати рейтинг постачальників в залежності від сумарного обсягу поставок для кожного типу продукції.
6	Білетна каса.
	1) Отримати інформацію на даний момент часу про всі рейсах, що виконуються в зазначений пункт призначення, дата і час прибуття яких в цей пункт знаходяться в зазначеному діапазоні значень. За кожним рейсом вивести також загальну кількість місць і кількість проданих квитків. 2) Для кожного рейсу здобути середню кількість зареєстрованих пасажирів за весь час експлуатації. Дані вивести в порядку убування значень. 3) Для кожного з типів транспортних засобів отримати його частку в загальному пасажиропотоку і сумарну кількість рейсів за тиждень. 4) Для кожного з пунктів призначення, які фігурують в розкладі, отримати їх частку в загальному пасажиропотоку. Впорядкувати по спадаючим значенням. 5) По кожній категорії квитків отримати сумарну вартість проданих квитків і їх частку від загальної суми виручки, отриманої від продажу квитків.
7	Відділ кадрів.
	1) Отримати список співробітників, що займають зазначену посаду, упорядкований за часом прийняття на роботу і ПІБ співробітників. 2) Отримати упорядкований по датах список співробітників, які пройшли підвищення кваліфікації. Для кожного співробітника вивести номер спеціальності і свідоцтва. 3) За кожним типом стягнення отримати загальну кількість накладених стягнень для всіх типів посад. 4) По кожному типу посади отримати загальну кількість заохочень за минулий рік. Дані впорядкувати по спадаючим значенням. 5) Отримати рейтинг співробітників за кількістю заохочень, стягнень і проходження курсів підвищення кваліфікації. Дані вивести в порядку убування рейтингу.
8	Готель.
	1) Отримати список клієнтів, які проживали в готелі в зазначений період. Список повинен бути впорядкований по тривалості проживання. 2) Отримати список клієнтів, які проживають в даний момент в готелі. Рядки в списку повинні бути згруповані за категоріями номерів і впорядковані за датою реєстрації. 3) Отримати дані про додаткові послуги, надані з початку поточного року, за категоріями номерів. Розміри додаткових послуг вивести в грошовому та відсотковому вираженні по відношенню до загальної суми оплати для даної категорії. 4) Отримати дані про кількість клієнтів, які проживали в готелі від 1 до 3, від 3 до 10 і понад 10 діб, в процентному вираженні від загальної кількості клієнтів. 5) Отримати дані про кількість клієнтів, які є іноземними громадянами, які проживали в готелі помісячно за минулий календарний рік.

Продовження таблиці 3.1

9	Виробництво.
	1) Отримати список продукції, що випускається на підприємстві по роках, упорядкований за собівартістю. 2) Отримати дані про заплановані на рік обсяги витрат сировини за типами продукції, що випускається. Дані впорядкувати за величиною витрат сировини при виготовленні. 3) За кожним типом продукції отримати дані про обсяги випуску з початку поточного року в процентному вираженні від плану, а також дані про кількість бракованої продукції в процентному вираженні від загального обсягу продукції даного виду. 4) Отримати обсяги поставок готової продукції та отриманий прибуток в грошовому вираженні для кожного виду продукції за минулий рік. 5) Отримати частку експорту в загальному обсязі виробленої продукції за всіма типами продукції.
10	Банк.
	1) Отримати списки клієнтів банку, юридичних і фізичних осіб, впорядковані за величиною залишків на рахунках. Кожен зі списків повинен починатися відповідним заголовком – «Юридичні особи» або «Фізичні особи». Вивести найменування для юридичних осіб або прізвище та ініціали для фізичних осіб. 2) Отримати список юридичних осіб, упорядкований за сумарними обсягами платежів, вироблених з початку поточного року. 3) За кожним видом платних послуг, що надаються банком фізичним особам, вивести загальну суму, отриману банком з початку поточного року. Отримати також підсумкове значення в цілому. 4) Отримати дані про кредити, видані фізичним особам. Значення вивести за кожним типом кредитів в процентному вираженні від загальної суми виданих кредитів. 5) По кожній юридичній особі отримати середню суму виданих кредитів. Дані вивести в порядку убування значень.
11	Експорт/Імпорт.
	1) Отримати дані про трьох підприємствах з найбільшим рейтингом за останні 5 балансових років. 2) За кожним кодом ТНЗД отримати сумарні обсяги експорту / імпорту за минулий рік. 3) Отримати середній обсяг експорту / імпорту за кожною ГВ. Результат впорядкувати по спадаючим значенням. 4) Отримати список підприємств, у яких чистий прибуток за минулий балансовий рік перевищив 50% валового доходу. 5) За кожним кодом ТНЗД отримати списки 3 підприємств з найбільшими рейтингами, які здійснювали експортно / імпортні операції за минулий балансовий рік.
12	Проект.
	1) Отримати дані про виконавців, для яких показники обсягу виконаних робіт та кількості використаних матеріалів не відповідають показникам найближчої за датою минулої віхи. 2) По кожному типу робіт отримати дані про поточний стан виконаного обсягу в процентному вираженні від загального запланованого обсягу і кількості використаних матеріалів в процентному вираженні від загальної потреби. 3) Отримати список «критичних робіт», тобто робіт, які не були завершені до зазначеної самої пізньої дати закінчення. Дані вивести за зменшенням затримки завершення. 4) Отримати загальну тривалість проекту в днях від моменту початку першої роботи (дата початку та найраніша дата початку збігаються) до моменту завершення останньої для неї (дата закінчення і найпізніша дата закінчення збігаються). 5) По кожному типу робіт отримати список виконавців, упорядкований по спадаючій запланованій частини загального обсягу даної роботи.

Вимоги до звіту

Звіт з лабораторної роботи повинен містити SQL-скрипти запитів контрольного завдання та завдання для самостійної роботи і результати їх виконання.

3.4 Контрольні питання

1) Поясніть призначення операторів BETWEEN, IN, LIKE.
2) Що таке агрегуючі функції і як вони використовуються?
3) Як називаються функції SUM, AVG, MAX, MIN, COUNT? Чому?
4) Чому агрегуючі функції не можуть застосовуватися одночасно з іменами полів під час відсутності модифікатора GROUP BY?
5) Що буде видано при використанні в списку вибору COUNT (*), COUNT (DISTINCT <Ім'я>)? Чому DISTINCT не можна застосовувати спільно з COUNT (*)?
6) Яке призначення пропозицій WHERE і HAVING?
7) Як можна вивести кілька даних різних типів в одному рядку символів?
8) Що таке декартовий добуток двох таблиць? Як його отримати?
9) Що таке еквісполучення двох таблиць? Як його отримати?
10) Що таке природне з'єднання двох таблиць? Як його отримати?
11) Що таке композиція двох таблиць? Як його отримати?
12) Для чого використовується об'єднання таблиці зі своєю копією?
13) Чому не можна використовувати оператори відношення для обробки результату простого вкладеного підзапиту, якщо він повертає більш одного значення?
14) Що таке корельований вкладений підзапит і як він обробляється?
15) У чому полягає схожість і відмінність підзапитів з ANY і ALL від підзапитів з EXISTS?
16) Які вимоги необхідно виконати для об'єднання двох запитів?
17) Що таке селекція і проекція таблиці?
18) Як засобами SQL виконати об'єднання, перетин і різницю двох таблиць?
19) Для чого призначена конструкція WITH?
20) Які особливості застосування пропозиції UNION?

Лабораторна робота №4

ВИЗНАЧЕННЯ ДАНИХ ТА МАНІПУЛЮВАННЯ ДАНИМИ

Створення та видалення бази даних. Створення та видалення базових таблиць. Інструкції мови маніпулювання даними

4.1 Теоретичні відомості

4.1.1 Інструкції мови визначення даних

4.1.1.1 Бази даних

Для створення баз даних в Transact-SQL служить інструкція CREATE DATABASE, синтаксис якої (зі скороченнями) має наступний вигляд:

```
CREATE DATABASE база_даних
[ [ WITH ] [ OWNER [=] користувач]
        /* власник бази даних (за замовчуванням – користувач, який ввів команду create)*/
[ TEMPLATE [=] шаблон ]
        /* шаблон, що використовується (за замовчуванням – template1) */
[ ENCODING [=] кодування ]
        /* кодування, що використовується (за замовчуванням – кодування шаблона) */
[ LC_COLLATE [=] порядок_порівняння]
        /* порядок порівняння рядків символів (за замовчуванням – як в шаблоні) */
[ LC_CTYPE [=] тип_категоризації]
    /* категоризація символів (нижній/верхній реєстр, цифра) (за замовчуванням – як в шаблоні) */
[ TABLESPACE [=] табличний_простір]
        /* табличний простір, що використовується (за замовчуванням – як в шаблоні) */
[ CONNECTION LIMIT [=] connlimit ] ]
      /* кількість паралельних підключень (за замовчуванням – «-1», тобто не обмежено) */
```

Наприклад, в результаті виконання команди:

```
CREATE DATABASE a_new_database
```

буде створена нова база даних з параметрами за замовчуванням:

```
WITH OWNER = postgres
ENCODING = 'UTF8'
TABLESPACE = pg_default
LC_COLLATE = 'Ukrainian_Ukraine.1251'
LC_CTYPE = 'Ukrainian_Ukraine.1251'
CONNECTION LIMIT = -1
```

Для внесення змін (зміна імені, власника, табличного простору, встановлення параметрів конфігурації) в існуючу базу даних служить команда:
ALTER DATABASE, а для видалення – DROP DATABASE:

Приклади:

```
/* створюємо базу даних BookShop_backup, використовуючи базу даних BookShop як шаблон(BookShop не
    повинна використовуватись) */
CREATE DATABASE BookShop_backup TEMPLATE "BookShop";
-- видаляємо базу данных BookShop_bacup
DROP DATABASE BookShop_backup;
```

4.1.1.2 Базові таблиці

Інструкція CREATE TABLE

Таблиці створюються за допомогою інструкції CREATE TABLE, яка в основному визначає тип таблиці (тимчасова або базова), ім'я таблиці, набір стовпців і обмеження цілісності (обмеження для стовпців або для всієї таблиці). Кожна таблиця повинна мати, щонайменше, один стовпець. Синтаксис інструкції CREATE TABLE в PostgreSQL в скороченому вигляді:

```
CREATE [ { TEMPORARY | TEMP } ] TABLE таблиця
(
            [ { стовпець типу_даних [ DEFAULT значення_за_замовчуванням]
   [<обмеження_стовпця> [ ... ] ]
     | < обмеження_таблиці >
     | LIKE батьківська_таблиця  [<параметри_ LIKE> ] } [, ... ] ]
)
[ INHERITS (батьківська _таблиця [, ... ] ) ]
[ TABLESPACE табличний_простір ]
```

де

```
< обмеження_стовпця > ::= [CONSTRAINT обмеження]
{  { NOT NULL | [ NULL ] }
 | CHECK (обмеження_на_значенння_стовпця)
 | UNIQUE параметри_індексування
 | PRIMARY KEY
      | REFERENCES таблиця [ (стовпець) ]
    [ MATCH FULL -- один стовпець в складеному зовнішньому ключі не може бути NULL
    | MATCH PARTIAL – не реалізовано
    | [ MATCH SIMPLE] ] -- один стовпець в складеному зовнішньому ключі може бути NULL
    [ ON DELETE дія ] -- дія при видаленні батьківського ключа
    [ ON UPDATE дія]  -- дія при зміні батьківського ключа
    }

< обмеження_таблиці > ::= [CONSTRAINT ім’я_обмеження ]
{ CHECK (обмеження_на_значення_стовпця)
 | UNIQUE (стовпець [, ... ] ) параметри_індексування
 | PRIMARY KEY (стовпець [, ... ] ) параметри_індексування
      | FOREIGN KEY (столбец [, ... ] ) REFERENCES таблиця [ (стовпець [, ... ] ) ]
    [ MATCH FULL | MATCH PARTIAL | MATCH SIMPLE ]
    [ ON DELETE дія]
    [ ON UPDATE дія] }
```

Якщо вказано TEMPORARY або TEMP, створюється тимчасова таблиця, тобто така таблиця автоматично видаляється відразу після завершення транзакції.

При створенні таблиць (або при їх зміні) можна вводити обмеження на значення окремих стовпців. Якщо обмеження введені, SQL буде відхиляти будь-які значення, які не задовольняють введеним обмеженням. Є два основних типи обмежень – **обмеження стовпця і обмеження таблиці**. Різниця між ними в тому, що обмеження стовпця застосовується тільки до індивідуальних стовпцями, в той час як обмеження таблиці застосовується до груп з одного або більше стовпців.

Обмеження NULL/NOT NULL дозволяють або забороняють введення в поле NULL-значень. Очевидно, що обмеження NOT NULL має бути зазначено для первинних

ключів, оскільки в іншому випадку під загрозою опиниться цілісність даних. Крім того, окремі поля таблиць за своїм призначенням можуть вимагати тільки певних значень. Якщо помістити ключові слова NOT NULL відразу після типу даних стовпця, будь-яка спроба помістити значення NULL в це поле буде відхилена. В іншому випадку, SQL буде вважати, що для цього стовпця NULL-значення дозволені.

Обмеження PRIMARY KEY (первинний ключ), використовується для призначення первинних ключів. Поля з таким обмеженням не можуть приймати NULL-значень, і навіть якщо обмеження NOT NULL не вказується для такого поля, SQL додає його за замовчуванням. Обмеження PRIMARY KEY автоматично вимагає унікальності введених даних. Будучи призначеним для декількох стовпців (складений первинний ключ) це обмеження ставить унікальність комбінацій відповідних значень, хоча окремо значення в кожному стовпці складеного ключа не обов'язково має бути унікальним.

Обмеження UNIQUE (унікальний), як і обмеження PRIMARY KEY, обмежує множину значень для зазначених стовпців унікальними значеннями. Як і обмеження PRIMARY KEY, обмеження UNIQUE може бути обмеженням таблиці, і тоді воно визначає унікальність комбінацій значень відповідних стовпців.

Обмеження FOREIGN KEY (Зовнішній ключ) забезпечує принцип посилальної цілісності. Коли стовпець є зовнішнім ключем, він певним чином пов'язаний з таблицею, на яку він посилається. Фактично це означає, що кожне значення в цьому стовпці (зовнішньому ключі) безпосередньо прив'язане до значення в іншому стовпці (батьківському ключі). Кожне значення (кожний рядок) зовнішнього ключа повинно недвозначно посилатися до одного і тільки цього значення (рядку) батьківського ключа. Якщо це так, то система буде в змозі забезпечувати посилальну цілісність. Зрозуміло, що будь-яке число зовнішніх ключів може посилатися до єдиного значенням батьківського ключа. Зрозуміло також, що в якості батьківського ключа можуть виступати стовпці з обмеженнями PRIMARY KEY або UNIQUE, тобто стовпці з унікальними значеннями і не містять NULL-значень. Зовнішній ключ може містити тільки ті значення, які фактично представлені в батьківському ключі або NULL-значення, спроба введення інших значень призводить до помилки. Присутність NULL-значень в зовнішньому ключі не порушить кількість посилань цілісність, але дасть можливість не вводити дані, якщо вони поки не відомі.

Подібно розглянутим вище обмеженням, обмеження FOREIGN KEY може бути обмеженням таблиці або стовпця. У першому випадку списки стовпців зовнішнього ключа і батьківських ключів, що перераховуються після службового слова REFERENCES і імені містить їх таблиці, повинні бути ідентичні. У другому випадку обмеження стосується лише одного стовпця і службові слова FOREIGN KEY не потрібні.

Обмеження CHECK дозволяє встановити умову, якій має задовольняти значення, що вводиться в таблицю, перш ніж воно буде прийнято. Обмеження CHECK складається з ключового слова CHECK, що супроводжується предикатом, який використовує вказане поле. Будь-яка спроба модифікувати або вставити значення поля, яке могло б зробити цей предикат невірним буде відхилена. Таким чином, можна запобігти введенню небажаних даних. Обмеження CHECK може використовуватися у вигляді деякої маски введення, що забезпечить контроль заданого формату даних, що вводяться.

Створювана таблиця може успадковувати стовпці інших таблиць, які задаються в реченні LIKE або INHERITS. Різниця між LIKE і INHERITS – в тому, що при використанні INHERITS створюється постійний зв'язок між дочірньою і батьківською таблицями, причому всі зміни, зроблені з батьківської таблицею, успадковуються дочірньою і за замовчуванням при вибірці значень батьківської таблиці дані з дочірньої автоматично включаються в цю вибірку. При використанні LIKE батьківська і дочірня таблиця повністю незалежні.

```
    <параметри_ LIKE> ::=
    { INCLUDING | EXCLUDING }  -- включаючи/виключаючи
{ DEFAULTS  -- значення за замовчуванням
```

| CONSTRAINTS -- *обмеження*
| INDEXES -- *індекси*
| STORAGE – *параметри збереження*
| COMMENTS -- *коментарі*
| ALL }

Інструкція CREATE TABLE AS

За допомогою інструкції CREATE TABLE AS можна створити таблицю, в тому числі тимчасову, на основі результатів запиту. Нова таблиця отримує ті ж стовпчики з тими ж типами даних, що і в запиті. При бажанні можна ввести інші імена стовпців для створюваної таблиці, а також зберегти дані, отримані в результаті виконання запиту, в створеній таблиці. Синтаксис інструкції CREATE TABLE AS (в спрощеному вигляді):

```
CREATE [ { TEMPORARY | TEMP } ] TABLE таблиця [ (стовпець [, ...] ) ]
    AS запит [ WITH [ NO ] DATA ]
```

Інструкція DROP TABLE

Базову таблицю можна в будь-який момент видалити за допомогою пропозиції DROP TABLE (знищити таблицю):

```
DROP TABLE [ IF EXISTS ] базова_таблиця [, ...] [ CASCADE | [RESTRICT] ]
```

по якому віддаляється опис таблиці, її дані, пов'язані з нею уявлення і індекси, побудовані для стовпців таблиці. Якщо вказано IF EXISTS і таблиці немає, то повідомлення про помилку не виводиться. Щоб видалити таблицю, на яку посилаються інші об'єкти бази даних, слід використовувати CASCADE, інакше операція відхиляється (за замовчуванням).

Для видалення вмісту таблиці (але не самої таблиці) можна використовувати інструкції DELETE або TRANCATE.

Інструкція ALTER TABLE

У SQL існує також інструкція ALTER TABLE (змінити таблицю), яка дозволяє додавати, видаляти і модифікувати стовпці і обмеження. Синтаксис ALTER TABLE має вигляд:

```
ALTER TABLE [ ONLY ] ім’я_таблиці [ * ]  <дія> [, ... ]
```

де (обмежуючись лише стандартними елементами)

```
<дія> ::= {
    -- додати стовпець
    ADD [ COLUMN ] стовпець тип  [обмеження_стовпця [ ... ] ]
    -- видалити стовпець
    | DROP [ COLUMN ] [ IF EXISTS ] стовпець [ RESTRICT | CASCADE ]
    -- змінити тип стовпця
    | ALTER [ COLUMN ] стовпець [ SET DATA ] TYPE тип
    -- змінити значення  за замовчуванням
    | ALTER [ COLUMN ] стовпець SET DEFAULT вираз
    | ALTER [ COLUMN ] стовпець DROP DEFAULT
    -- змінити обмеження NOT NULL
    | ALTER [ COLUMN ] стовпець { SET | DROP } NOT NULL
    -- перейменувати стовпець
    | RENAME [ COLUMN ] ім’я_стовпця TO нове_ім’я
    -- додати/видалити обмеження таблиці
    | ADD обмеження_таблиці
    | DROP CONSTRAINT [ IF EXISTS ]  обмеження [ RESTRICT | CASCADE ] }
```

Однак модифікація таблиці за допомогою ALTER TABLE може змінити права доступу інших користувачів до цієї таблиці, тому часто буває простіше просто створити нову таблицю

з необхідними змінами при створенні, і переписати в неї дані зі старої таблиці.

Приклади:

```
/* створюємо нову таблицю Доставка */
CREATE TABLE "Доставка"
( "Замовлення" integer,
  "Кур'єр" character varying(40) DEFAULT 'foo',
  "Дата_час" timestamp without time zone DEFAULT CURRENT_TIMESTAMP,
  "Доставлений" boolean DEFAULT FALSE);

/* додаємо первинний ключ*/
ALTER TABLE "Доставка" ADD CONSTRAINT "pk_доставка"
  PRIMARY KEY ("Замовлення");
/* додаємо зовнішній ключ для посилання на Замовлення*/,
ALTER TABLE "Доставка" ADD CONSTRAINT "fk_доставка"
FOREIGN KEY ("Замовлення") REFERENCES "Замовлення" ("Код_замовлення")
MATCH SIMPLE  ON UPDATE CASCADE ON DELETE SET NULL;
```

4.1.2 Інструкції мови маніпулювання даними

4.1.2.1 Інструкція INSERT

Таблиці створюються інструкцією CREATE TABLE. Ця інструкція створює порожню таблицю – таблицю без рядків. Значення вводяться, видаляються або оновлюються за допомогою інструкцій **мови маніпулювання даними** (DML), основними з яких є інструкції INSERT (вставити), DELETE (видалити), і UPDATE (оновити). Подібно інструкції SELECT вони можуть оперувати як базовими таблицями, так і представленнями.

Синтаксис інструкції INSERT:

```
INSERT [INTO] {базова_таблиця | представлення} [(стовпець [, ...] )]
{  DEFAULT VALUES
    | VALUES ({ DEFAULT | NULL | вираз } [ , ...] )
    | запрос }
[ RETURNING { * | выражение [ [ AS ] ім'я ] } [, ...] ]
```

Фраза INTO не обов'язково і просто покращує читабельність інструкції.

Фраза RETURNING (нестандартне) дозволяє вивести на екран (також, як і при використанні SELECT) значення виразів, побудованих на основі даних, що додаються.

Можливі два варіанти використання INSERT: без запиту і з запитом. У першому варіанті в зазначену таблицю вставляється рядок зі значеннями полів, зазначеними в переліку пропозиції VALUES (значення), причому і-е значення має відповідати і-му стовпцю в списку стовпців (як і в інструкції SELECT, порядок слідування стовпців може бути довільним). Стовпці, які не вказані в списку, заповнюються NULL-значеннями або значеннями за замовчуванням. Якщо NULL-значення для такого стовпчика не припустимі і не вказано значення за замовчуванням, то транзакція скасовується і виводиться повідомлення про помилку. Якщо в списку VALUES вказані всі стовпці таблиці і порядок їх перерахування відповідає порядку стовпців в описі таблиці, то список стовпців після імені таблиці або представлення можна опустити.

У другому варіанті використання інструкції INSERT спочатку виконується запит, тобто за пропозицією SELECT в пам'яті формується робоча таблиця, а потім рядки робочої таблиці завантажуються в INSERT-таблицю. При цьому і-й стовпець робочої таблиці (і-й елемент списку SELECT) відповідає і-му стовпцю в списку стовпців таблиці, що модифікується. Тут також при виконанні зазначених вище умов може бути опущений список стовпців фрази

INTO. Інструкція INSERT із запитом частіше застосовується для перенесення рядків з однієї таблиці в іншу (можливо через іншу таблицю або представлення).

Приклади:

```
/* вставляємо в таблицю Доставка новий рядок */
INSERT INTO "Доставка"
VALUES (  1000,                  -- Замовлення
          DEFAULT,               -- Кур'єр
          DEFAULT,               -- Дата_час
          DEFAULT)               -- Доставлений

/* вставляємо в таблицю Доставка всі сплачені замовлення */
INSERT INTO "Доставка" ("Замовлення", "Дата_час")
SELECT "Код_замовлення", "Дата" + '1 0:0'::interval
FROM  "Замовлення"
WHERE "Сплачено" LIKE '%так%';

/* створюємо таблицю Просрочені_ доставки  за структурою таблиці Доставка
CREATE TABLE "Просрочені_ доставки" (LIKE "Доставка")

/* вставляємо в таблицю Просрочені_ доставки всі просрочені доставки */
INSERT INTO "Просрочені_ доставки"
SELECT *
FROM  "Доставки"
WHERE NOT("Доставлений") AND  "Дата_час" < now();
```

4.1.2.2 Інструкція DELETE

Інструкція DELETE дозволяє видалити вміст всіх рядків зазначеної таблиці або тих її рядків, які задовольняють зазначеній умові. Ця інструкція має формат:

```
DELETE FROM [ ONLY ] {базова_таблиця | представлення}  [ [ AS ] псевдонім ]
[ USING список ]
[ WHERE { умова | WHERE CURRENT OF курсор } ]
[ RETURNING { * | вираз [ [ AS ] ім'я ] [, ...] } ]
```

Список в реченні USING аналогічний списку фрази FROM інструкції SELECT.

При відсутності фрази WHERE видаляється вміст всіх рядків зазначеної таблиці, в іншому випадку – ті її рядки, які задовольняють умові в фразі WHERE.

Фраза RETURNING аналогічна такій же фразі інструкції INSERT.

Перед видаленням рядків з таблиці має сенс виконати інструкцію SELECT з такою же умовою відбору рядків, як і умова в реченні WHERE інструкції DELETE, з тим, щоб переконатися в коректності виконуваної операції видалення.

Для видалення всіх рядків інструкція DELETE використовується рідко. Якщо необхідно видалити всі рядки в таблиці, у багатьох випадках слід віддати перевагу інструкції TRUNCATE TABLE, тому що ця інструкція працює більш швидко, видаляючи посилання даної таблиці на фізичні сторінки даних, а не видаляючи дані через підрядник як інструкція DELETE. Але слід пам'ятати, що TRUNCATE TABLE, на відміну від DELETE, не записує видалені рядки в журнал транзакцій, а значить, видалені рядки відновити не можна. Крім того, TRUNCATE TABLE не може бути застосована до таблиць, на які посилаються інші таблиці через зовнішні ключі.

Приклади:

/ видаляємо з таблиці Замовлення рядки з невизначеними значеннями коду замовника */:*

```
DELETE FROM “Замовлення”
WHERE “Код_замовника” IS NULL;
```

/ видаляємо з Просрочені_ доставки рядки, що відповідають невизначеним замовникам */*

```
SELECT  "Код_заказа" FROM "Замовлення"  WHERE "."Код_замовника" IS NULL;
DELETE FROM "Просрочені_ доставки"
USING "Замовлення"
WHERE ("Замовлення" = "Замовлення"."Код_замовлення") AND
            ("Замовлення"."Код_замовлення" IS NULL);
```

4.1.2.3 Інструкція UPDATE

Інструкція UPDATE використовується для зміни існуючих значень. У різних діалектах SQL формати цієї інструкції можуть відрізнятися як один від одного, так і від формату ANSI SQL. У PostgreSQL прийнятий наступний формат (в спрощеному вигляді):

```
UPDATE [ ONLY ] {базова_таблиця | представлення}  [ [ AS ] псевдонім ]
SET { стовпець = { вираз | DEFAULT } |
      (стовпець [, ...] ) = ( { вираз | DEFAULT } [, ...] ) } [, ...]
[ FROM список ]
[ WHERE умова | WHERE CURRENT OF курсор ]
[ RETURNING { * | вираз [ [ AS ] ім’я ] [, ...] } ]
```

За відсутності необов'язкових фраз FROM і WHERE оновлюються значення зазначених в списку фрази SET стовпців у всіх рядках таблиці, що модифікується. Але список фрази SET може включати лише стовпці з оновлюваної таблиці, тобто значення одного з стовпців оновлюваної таблиці може замінюватися на значення її іншого стовпця чи виразу, що містить значення декількох її стовпців, включаючи стовпець, що змінюється.

Пропозиція WHERE дозволяє скоротити число оновлюваних рядків, вказуючи умови їх відбору.

Пропозиція FROM дозволяє проводити оновлення значень оновлюваної таблиці за значеннями стовпців з інших таблиць, представлень чи підзапитів. Перелік таблиць-джерел фрази FROM містить ім'я оновлюваної таблиці і тих таблиць, значення стовпців яких використовуються для оновлення. При цьому, природно, таблиці повинні бути пов'язані один з одним через фразу WHERE. У виразах, значення яких присвоюються стовпцями в списку пропозиції SET, слід уточнювати імена використовуваних стовпців, випереджаючи їх ім'ям таблиці або псевдонімом, при цьому в якості виразу може виступати підзапит, який повертає скалярне значення.

Якщо команди маніпулювання даними порушують обмеження, встановлені на стовпці таблиць, то відповідні транзакції скасовуються, і виводиться повідомлення про помилку. Найбільш важливим є питання про взаємовідносини команд маніпулювання даними та зовнішніх і батьківських ключів. Для стовпців, визначених як зовнішні ключі, будь-які значення, які поміщаються в ці стовпці командами INSERT або UPDATE повинні бути представлені в їх батьківських ключах. Можна поміщати NULL-значення в ці стовпці, незважаючи на те, що NULL-значення не припустимі в батьківських ключах. Можна видаляти або змінювати будь-які рядки з зовнішніми ключами, але будь-яке значення батьківського ключа не може бути видалено або змінено. Це означає, наприклад, що не можна видалити запис про замовника з таблиці Замовники поки на цю таблицю є посилання зовнішніх ключів інших таблиць.

Для вирішення аномалій, що виникають при зміні або видаленні поточного значення батьківського ключа, кажучи в цілому про SQL, є три можливості:

– Заборонити зміни в батьківському ключі (обмежена зміна – RESTRICTED).

– Дозволити автоматичну зміну зовнішнього ключа (каскадна зміна – CASCADES).

– Дозволити зміну в батьківському ключі і автоматично встановити зовнішній ключ в NULL (SET NULL).

Приклади:

```
/* коректуємо  кількість  книг по замовленню з врахуванням наявного залишку */
UPDATE  "Замовлення"
SET "Кількість" = T2. "Залишок"
FROM  "Замовлення" T1, "Книги" T2
WHERE T1. "Код_книги" = T2."Код_книги ” AND T1. "Кількість" > T2. "Залишок"
```

4.2 Завдання для самостійної роботи

1) Створіть таблицю Співробітники в базі даних BookSHOP. Таблиця повинна містити поля: персональний_номер (первинний ключ), прізвище_ім’я, розмір_комісійних (значення за замовчуванням – 3%), посаду («директор», «консультант», «молодший продавець», «старший продавець», «закупник»), дата_прийому_на_роботу (за замовчуванням – поточна дата).
2) У таблицю Замовлення внесіть наступні зміни: а) додайте стовпець Оформив_замовлення з обмеженнями NOT NULL і зовнішнього ключа з посиланням на стовпець Персональний_номер таблиці Співробітники; б) додайте стовпець Кількість з обмеженням NOT NULL і значенням за замовчуванням – 1.
3) Змініть таблицю Співробітники так, щоб розмір комісійних за замовчуванням дорівнював 5% і для цього поля не допускалися NULL-значення.
4) У таблиці Замовлення до складу первинного ключа додатково введіть поле Код_книги (Вказівка: для цього слід видалити наявний в таблиці первинний ключ)
5) Напишіть інструкцію, результатом виконання якої була б вставка декількох рядків (окремо та разом) в таблицю Співробітники, створену раніше.
6) Напишіть інструкцію, результатом виконання якої було б видалення всіх неоплачених на поточну дату замовлень з таблиці Замовлення.
7) Напишіть інструкцію, результатом виконання якої було б збільшення комісійних трьох співробітників, що мають найбільшу кількість оформлених замовлень.
8) Створіть таблицю такої ж структури, що і таблиця Співробітники, і напишіть інструкцію для вставки в неї даних про всіх співробітників з таблиці Співробітники, які оформили більше трьох замовлень і мають стаж не менше 5 років.
9) Напишіть інструкцію, результатом виконання якої було б видалення з таблиці Замовники тих замовників, які зробили останнє замовлення більше 1 року тому. Видаліть відповідні замовлення з таблиці Замовлення.
10) У таблицю Замовлення додайте стовпець Номер_договору типу uuid, введіть в кожному рядку в цьому стовпці значення, яке дорівнює кількості секунд, що пройшли від '01 .01.1900 00:00 ' до того моменту, коли чергове таке значення вноситься в таблицю, і зробіть Номер_договору первинним ключем, попередньо видаливши створений раніше первинний ключ.

4.3 Контрольне завдання

1) Створити копію розробленої раніше бази даних, включаючи вміст таблиць. Всі подальші операції виконувати для цієї копії.

2) В одну з базових таблиць, на яку не посилаються зовнішні ключі інших таблиць, додати стовпець типу uuid і зробити його первинним ключем, скасувавши існуюче у цій таблиці обмеження первинного ключа. В цей стовпець в кожному рядку ввести значення, яке дорівнює кількості секунд, що пройшли від '01 .01.1900 00:00' до того моменту, коли чергове таке значення вноситься в таблицю.

3) Напишіть інструкцію, результатом виконання якої була б вставка декількох рядків (окремо та разом) в одну з базових таблиць.

4) Напишіть інструкції для внесення даних в усі обчислювані стовпці базових таблиць на основі значень цієї та інших таблиць (якщо таких стовпців немає, створити 2-3 таких стовпчика за вказівкою викладача).

5) Напишіть 2-3 інструкцій для видалення строк з однієї базових таблиць за деякою умовою, в якій фігурують значення як цієї так і інших таблиць.

6) Створіть таблицю такої ж структури, що і одна з базових таблиць, і напишіть інструкцію для вставки в неї даних з базової таблиці, що задовольняють деякій умові.

7) В одній з базових таблиць змініть тип обмеження зовнішнього ключа. Якщо така зміна не можлива, виконайте операції видалення та вставки обмеження з новим типом.

8) Оформити звіт про виконання контрольного завдання. Звіт з лабораторної роботи повинен містити:

- звіти DDL для всіх базових таблиць розробленої БД;
- скрипти і результати виконання запитів контрольного завдання.

4.4 Контрольні питання

1) Дайте характеристику основних типів обмежень.
2) Як можна забезпечити унікальність значень для даного стовпця або групи стовпців?
3) Що буде введено в стовпець, якщо в інструкції INSERT цей стовпець не вказано?
4) Чим відрізняється TRUNCATE від DELETE?
5) Як в SQL вирішуються аномалії видалення і оновлення значень батьківських ключів?

РЕКОМЕНДОВАНА ЛІТЕРАТУРА

Базова

1. Дейт К. Введение в системы баз данных: пер. с англ. / К.Дж. Дейт 8-е издание – М.: Вильямс, 2006. – 1326 с.
2. Ульман Д. Введение в системы баз данных: пер. с англ. / Д. Ульман, Д. Уидом. – М: Лори, 2000. – 512 с.
3. Колби Дж. SQL для начинающих: пер. с англ. / Дж. Колби, П. Уилтон. – М.: Вильямс, 2006. – 496 с.
4. Кевин Кл. SQL: справочник: пер. с англ. / Кл. Кевин, 2-е издание. – М: Кудиц-Образ, 2006. – 832 с.
5. Мюллер Р. Базы данных и UML. Проектирование: пер. с англ. / Роберт Дж. Мюллер – М: Лори, 2002. – 420 с.

Допоміжна

1. Simon R. PostgreSQL 9 Administration Cookbook / Simon Riggs, Hannu Krosing – PacktPublishing, 2010. – 336 p.
2. Дюбуа П. MySQL: пер. с англ. / П. Дюбуа. – М.: Вильямс, 2001. – 236 с.
3. Бен-Ган И. Microsoft SQL Server 2008. Основы T-SQL: пер. с англ. / Ицик Бен-Ган. – С.-П.: БХВ-Петербург, 2009. – 430 с.

Інформаційні ресурси

1. MySQL: The world's most popular open source database [Електронний ресурс]. – Режим доступу: URL: http://www.mysql.com/. – Назва з екрану.
2. PostgreSQL: The world’s most advanced open source database [Електронний ресурс]. – Режим доступу: URL: http://www.postgresql.org/. – Назва з екрану.
3. SQL Server 2008 R2 Books Online [Електронний ресурс]. – Режим доступу: URL: http://msdn.microsoft.com/en-us/library/ms167593(v=sql.105).aspx. – Назва з екрану.
4. Обучающее руководство по PostgreSQL [Електронний ресурс]. – Режим доступу: http://ods.com.ua/koi/db/postgres/tutorial/tutorial.html. – Назва з екрану.