Изпитна тема № 6: Бази данни

Въведение в базите от данни. Типове данни. Основни команди. Основни SQL команди за работа бази данни – създаване, промяна и премахване на бази от данни, таблици и колони. Моделиране на релационни бази от данни – нормализация и нормални форми. Типове връзки, каскадни операции и ограничения. Заявки за извличане и промяна на данни (SELECT, INSERT, UPDATE, DELETE). Сложни заявки за извличане на данни. Съединения на таблици (SQL JOIN). Агрегиращи функции и групиране на данни. Филтриране и сортиране на групирани данни. Функции, транзакции и съхранени процедури.

Базата данни е инструмент за събиране и организиране на информация. Базите данни могат да съхраняват информация за хората, продуктите, поръчките или каквото и да било друго. Много бази данни се стартират като списък в програма за текстообработка или електронна таблица. Тъй като списъкът става по-голям, съкращенията и несъответствията започват да се показват в данните.

SQL (Structured Query Language) е език за структурирани заявки.Това не е език за програмиране, а език за заявки който позволяват създаването на релационни бази данни и работа с представената в тях информация. SQL стандартът е определен от Американския национнален институт по стандартизация (ANSI Americal National Standards Institute ). Също така той е възприет от международната организация по стандартизация (ISO - International Organization for Standartization) в качеството на международен стандарт.

След като релационните бази от данни станаха популярни, възникна нуждата от стандартен език за операции с данни. Отговорът бе SQL (Structured Query Language, Език за структурирани заявки). Постепенно SQL прерасна в многофункционален език за работа с бази данни с управляващи конструкции за: създаване, промяна и изтриване на данни; дефиниране на данни (таблици, колони); защита на достъпа до елементи от бази данни чрез работа с групи и индивидуални потребители; операции за управление на данните като създаване на архивни копия, блоково копиране и актуализация; и, най-важното, обработка на транзакции.

Под релация в теорията на БД се има предвид множество от записи, които имат едни и същи атрибути. Съществуват определени правила, на които трябва да отговаря една база данни и те са:

* Всяка релация трябва да има уникално име в рамките на една база данни.
* Всеки атрибут трябва да има уникално име в рамките на една релация.
* Недопустимо е съдържанието на два еднакви реда в една релация.
* Стойностите на всички записа са атомарни – състоят се само от един тип данни и не трябва да са резултат от изчисление.
* Записите в една релация нямат строга подредба.
* Всяка релация трябва да има поне един уникален идентификатор(атрибут/колона), наречен първичен ключ, по който всеки един ред в релацията да може да бъде еднозначно идентифициран.

Всяко поле си има свой тип данни. Типът данни определя какви данни могат да се съхраняват в това поле. Всички стойности в това поле трябва да имат един и същи тип.

Числови типове(Numeric Data Types):

INT(L)– Цяло число, може да бъде signed или За signed стойностите му варират в интервала от -2147483648 до 2147483647. За unsigned: от 0 до 4294967295. Може да специфицирате дължина до 11 цифри.

BIT(L) – приема две стойности – 0 или 1, ако не му се специфицира L(length). В противен случай приема до 64 бита.

TINYINT(L) – Много малко цяло число, което отново може да бъде signed или unsigned. За signed стойностите му варират от 32768 до 32767. За unsigned са от 0 до 65535. Можете да специфицирате дължина до 5 цифри.

MEDIUMINT(L) – Средно по големина цяло число, което отново може да бъде signed или unsigned. За signed стойностите му варират в интервала от -8388608 до 8388607. За unsigned са от 0 до 16777215. Можете да специфицирате дължина до 9 цифри.

BIGINT(L) – Голямо цяло число, което отново може да бъде signed или unsigned. За signed стойностите му варират в интервала от -9223372036854775808 до 9223372036854775807. За unsigned са от 0 до 18446744073709551615. Можете да специфицирате дължина до 20 цифри.

BOOL, BOOLEAN –Булев(логически) тип. Синоними е на TINYINT(1). Работи със стойности 0 (false) или която и да е друга цифра (true).

FLOAT(M,D) – Число с плаваща запетая, което не може да бъде беззнаково. Можете да дефинирате дължина на числото(M), както и брой числа след запетаята(D). И двата атрибута не са задължителни. Стойностите на M и на D по подразбиране са 10 и 2. Тоест 10 е броя на цифрите на числото, а 2 – броят на цифрите след десетичната запетая. Точността може да варира до максимум 24 знака след запетаята.

DOUBLE(M,D) – Число с плаваща запетая с двойна точност и не може да бъде беззнаково. Можете да дефинирате дължина на числото(M), както и брой числа след запетаята(D). И двата атрибута не са задължителни. Стойностите на M и на D по подразбиране са 16 и 4. Точността може да варива до 54 знака след запетаята. Типът REAL е синоним на DOUBLE.

DECIMAL(М, D) – Число с плаваща запетая с двойна точност, като DOUBLE, но е записано по такъв начин, че различните клиенти на БД ще прочетат числото с неговата оригинална точност . При DOUBLE, например, винаги съществува опасност от загуба на точност от закръгляне. За сметка на това предимство, работата с DECIMAL е по-бавна отколкото с DOUBLE.

NUMERAL(М, D), DEC(M,D), FIXED(M,D) –всички са синоними на DECIMAL.

Текстови типове(String Types):

CHAR(L) – Символен низ с фиксирана дължина от 1 до 255 символа. Параметърът L не е задължителен. По подразбиране стойността на L е 1. Колкото и символи да запишете под L, винаги ще се заделя памет за L символа.

VARCHAR(L) – Символен низ с променлива дължина. Параметърът L е задължителен. Заема толкова памет, колкото е нужно за съхранение на подадения низ. Теоретично заема 1 до 65535 символа, но от практична гледна точка може да заеме такава дължина, колкото е дължината на един ред в MySQL (общият сбор от байтове, които заемат всички колони). В по-стари версии на MySQL максималната дължина е 255.

BLOB или TEXT – текстови полета с максимална дължина от 65535 символа. BLOB – Binary Large Objects –обикновено се използва за съхранение на голям брой байтове, като например изображения. Всички колони, декларирани с тип TEXT обикновено се използват за съхранение на големи по обем текстове. Разликата между двата типа е. че при сортиране и при сравнения на данни при BLOB няма значение Case Sensitive, a при TEXT има. Не се специфицира дължина. TEXT полетата работят с референции към истинските стойности. Тоест в самата таблица се съхраняват указатели към реалното местоположение на символните низове. Предимство е по-малкия обем на дадената таблица, недостатък е по-бавното извличане. BLOB полетата съхраняват байтове вместо символи, както е при

TINYBLOB или TINYTEXT – Символни низове с максимална дължина до 255 символа. Без специфициране на дължина.

MEDIUMBLOB или MEDIUMTEXT – Символни низове с максимална дължина до 16777215 знака.

LONGBLOB или LONGTEXT – Символни низове с максимална дължина до 4294967295.

BINARY(bytes) – Подобно на CHAR, но работи директно с битове. Ако се налага да се сравняват такива полета, то това сравнение ще трябва да е побитово. На практика работи много по-бързо от CHAR.

ENUM() – Изброим тип данни. Например ENUM (‘X’, ‘Y’, ‘Z’). В скобите се подава списък с възможните стойности на полето. Може да бъде и NULL. Стойностите могат да бъдат максимум 65535.

SET() – Изброим тип данни, подобен на ENUM, но с тази разлика, че позволява в една клетка от таблицата да бъдат записани повече от една от изброените стойности.

Типове за дата и час:

DATE – дата във формат YYYY-MM-DD. Стойностите са между 1000-01-01 и 9999-12-31. Например датата 14 Февруари 1945 ще бъде 1945-02-14.

DATETIME – Дата и час във формата YYYY-MM-DD HH:MM:SS.

TIMESTAMP – Дата и час във формат Форматът поддържа дата и час от 1 януари 1970 до 2037 година. Няма никакви разделители между отделните компоненти на датата.

TIME – време, съхранено във формат HH:MM:SS.

YEAR(M) – година, съхранена в двуцифрено или в четирицифрено число в зависимост от параметъра За параметър 2 годините могат да бъдат от 1970 до 2069 като числата от 70 до 99 представляват годините от 1970 до 1999, а числата от 0 до 69 представляват годините от 2000 до 2069. За параметър 4 – възможните са от 1901 до 2155 година. По подразбиране стойността на М е 4. Параметър М със стойност 2 вече е deprecated.

Класовете в ER-диаграмата се превръщат в таблици, а техните атрибути– в атрибути на таблиците.

Характеризиращите обекти са обикновени таблици, с външен ключ към таблицата, която характеризират.

Подклас-обектите са си таблици, свързани с таблиците на основните им обекти обикновено чрез връзка 1:1, като в зависимост от спецификата на предметната област е възможно и свързване 1:М.

Връзка от тип 1:1 между две таблици се осъществява с налагане ограничение UNIQUE(забранява повтарящи се стойности) върху колона външен ключ във втората таблица, която сочи към първичния ключ от първата.

Връзка тип 1:М между две таблици се осъществява като колона от втората таблица стане външен ключ, сочещ към първичния на първата таблица.

Връзка тип М:М между две таблици с реализира с помощта на трета допълнителна (свързваща) таблица, в която атрибутите са два външни ключа, сочещи съответно към свързваните таблици.

Създаване на база данни:

SQL е разделен най-общо в две направления – структурна дефиниция и манипулация над данните. Затова езикът условно се разделя на DDL(Data Definition Language) и DML(Data Manipulation Language).

DDL– съдържа команди, с които се дефинира структурата на базата данни – схеми, таблици, ограничения и връзки между таблиците. Това са всички заявки от типа:

CREATE DATABASE/ CREATE TABLE – създаване на база данни/ създаване на таблица. Пример:

CREATE TABLE <име на таблицата>(

колона1 <тип данни> <…ограничения…>,

колона2 <тип данни> <…ограничения…>,

колона3 <тип данни> <…ограничения…>

);

ALTER DATABASE/ALTER TABLE – промяна дефиницията на база/ на таблица. Синтаксис на ALTER TABLE заявки: Понякога се налага да добавите нови колони, да модифицирате стари или да ги изтриете от дадена таблица. Също така не е възможно да създадете външен ключ към колона, която не сте декларирали предварително. Затова често външния ключ се добавя по-късно. Става с командата ALTER TABLE. Например:

ALTER TABLE <име на табл.>

ADD колона1 <тип данни> <…ограничения…>

За промяна на текуща колона :

CHANGE <име на колона> <ново име на колона> <тип данни> <ограничения>

Също така може да използваме DROP – за изтриване:

DROP <име на колона>.

Ограничения:

Ограниченията в езика SQL правят проверка за валидност на данните в определена колона.

NULL/NOT NULL – разрешава/забранява празни стойности

UNIQUE – задължава една колона да има уникални стойности (връзка 1:1)

DEFAULT – задава стойност по подразбиране.

Например:

group INT DEFAULT 1; -задава стойност по подразбиране 1.

CHECK – В повечето езици означава проверка за определена стойност.

Заявки INSERT:

Заявките INSERT, както им подсказва името, се използват за вмъкване на данни в таблица. Синтаксис:

INSERT INTO <таблица>(колона1,колона2,колона N,…)

VALUES(стойност1,стойност2,стойностN,…);

Заявки UPDATE. Те позволяват обновяване (редактиране) на стойностите на колоните за вече направени записи. Синтаксис:

UPDATE <таблица>

SET <колона1> = <нова стойност>,

<колона2> = <нова стойност>

WHERE <условие>;

Заявки DELETE:

За да изтрием съществуващ ред използваме следния синтаксис:

DELETE FROM <таблица>

WHERE <условие>;

JOIN заявки (съединения на таблици) - JOIN заявките се разделят най- общо на вътрешни и външни съединения. За да може правилно да свържем двете таблици, ще ни трябва някакъв критерий, на базата на който да съединим ред от едната с ред от другата. Ще го наричаме свързващо условие.

INNER JOIN

Такова съединение между две релации, при което се получава резултатна таблица, съставена от колони от двете, базирана на свързващо условие. Заявката сравнява всеки ред от първата с всеки ред от втората за да открие всички двойки, които отговарят на свързващото условие. Получава се нещо подобно:

SELECT <табл1.колона1>,<табл2.колона2>

FROM <табл1> JOIN <табл2>

ON <свързващо условие>

WHERE <ограничаващо условие>;

Условието в ON ще наричаме свързващо. То трябва да е изпълнено и за двете таблици.

Вложен SELECT:

При таблици между които няма пряка връзка се налага да се „минава“ през една или друга таблица за да се стигне до трета.

Имаме две условия:

ON – свързващо

WHERE – ограничаващо

Не сме ограничени да ги разменим или да ги обединим с логическо AND по между им и заявката ще работи по същия начин. Реално между двете условия е приложено логическо И.

Добра практика е в ON да слагаме само свързващи условия, а всички останали – в ограничаващи.

Външни съединения – OUTER JOIN:

Външните съединения се делят на две – LEFT и RIGHT OUTER JOIN. Най-общо казано, те ни дават всички редове от лявата/дясната таблица, без значение от това дали има съответстващи им редове от дясната/лявата таблица. Може да се каже, че при ляво съединение „натежава“ лявата таблица, а при дясно – дясната. Думата OUTER не е задължителна, така, както и INNER.

Често се налага да обединим резултата от една или повече SELECT заявки в една резултатна таблица. Това може да стане като използваме заявка за обединение – UNION.

(SELECT a…..)

UNION или UNION ALL

(SELECT b…..);

Резултатът ще бъде обединението от двата SELECT-а.

!!! Важно:

* Броят на колоните в двете заявки трябва да бъде един и същ!
* Не е задължително типът на тези колони да е един и същ!
* За имена на колони на резултатната таблица се взимат имената на колоните от първата заявка!
* В резултата няма да присъстват повтарящи се редове ( работи на принципа на SELECT DISTINCT –резултатите са без повторения).
* Ако желаете да получите всички редове, включително и повтарящите се, може да използвате UNION ALL.

Агрегатни функции:

Агрегатните функции ни позволят да правим обобщения по множество от стойности, да търсим мин. или макс. елемент в такова множество и др. А те са: SUM(); MAX(); MIN(); AVG(); COUNT(). В скобите обикновено присъства име на колона, за която желаем да бъде приложена агрегатната функция.

Групиране на данни – GROUP BY: Агрегатните функции работят с цялото множество от стойности на колоните, които са им подадени и съответно изчисялват и връщат един резултат. Как да накараме, обаче, една агрегатна функция да изчислява не цялото множество, а определени групички от него. За целта използваме клаузата GROUP BY. Обикновено групирането се използва в комбинация с агрегатни функции. Целта е агрегатната функция да се приложи върху множество по-малки резултатни таблици или групи, които накрая ще се обединят в една обща.

Клауза HAVING: Използва се за финално филтриране на данните в резултата, след изпълнението на агрегатната функция. Условията, които касаят резултат от агрегатна функция, се слагат в HAVING, другите ограничаващи условия може да се сложат също в HAVING, но не е добра практика, тъй като те ще се проверят и ще ограничат само и единствено крайната резултатна таблица и по време на самата заявка агрегатната функция ще работи с пълното множество от данни, съединенията ще се правят между пълни множества и т.н. Затова ползвайте правилото – ограничаващите условия в WHERE, а всички, които касаят резултат от функция – в HAVING.

Трансакция наричаме последователност от SQL заявки, които трябва да изпълняват условието: или всичките да бъдат изпълнени или нито една от тях да не бъде изпълнена. Трансакциите гарантират, че ако някоя заявка не се изпълни, то данните ще бъдат възстановени в първоначалния им вид. Синтаксис: Единствено трябва да оградим данните с BEGIN (започване на трансакция) и с COMMIT (край на трансакция). Ако някоя от заявките пропадне, то се прави т.нар. ROLLBACK. За целта sql сървъра използва innodb log файл, в който се записват старите данни, преди изпълнението на всяка заявка. Когато един или повече редове се модифицират в рамките на незавършена транзакция, то те биват заключвани докато не се подаде команда COMMIT или ROLLBACK. Всяка друга заявка, опитваща се да достъпи такива заключени редове, моментално попада в т.нар. „спящ“ (idle-режим). Съществува определен timeout, след който заявката се анулира, ако съответните редове все още не са отключени и транзакцията прави rollback. Заключването на връзките важи само за заявки от тип UPDATE, INSERT и DELETE. По подразбиране то не е валидно за заявки от тип SELECT, но може и да бъде променено.

Съхранените процедури представляват съвкупност от SQL заявки, съхранени в специален каталог в базата данни и много наподобяват методите(функциите) в езиците за програмиране. Често се налага да се пазят заявки, които ще се ползват често, но с различни параметри (динамични), като например отчети, справки, проверки и тн. Друг типичен случай е, когато се налага да правим зареждане на данни от едно място на друго, или когато трябва да изпълним последователност от операции за приключване на отчетен период, например. Тогава удобно решение е да имаме процедури в базата, съдържащи необходимото множество от заявки, които да бъдат извиквани от някаква програма или от самия сървър в определено време. Писането на съхранени процедури може да увеличи производителността на едно приложение. Веднъж написани, процедурите в MySQL се компилират всеки път по време на извикването им и компилираната им версия се пази в кеша на сървъра за текущата отделна връзка (сесия). Съществено предимство на работата с процедури е, че те намаляват значително трафика на данни между приложението и сървъра на СУБД. Вместо да се изпращат множество заявки към сървъра, той да връща отговори на всяка една от тях, програмата да проверява или валидира тези отговори и пак да върне заявка към сървъра, то всичко това би могло да се случи само с извикването на името на процедурата и предаването на определени параметри, с които тя да си свърши работата.