RDBMS

"Система управления
реляционными базами данных"

Общие понятия

- **SQL** означает "Структурированный Язык Запросов" (Structured Query Language)
- **RDBMS** - "Система управления реляционными базами данных" (Relational Database Management System)

Реляционная база данных представляет собой набор таблиц (сущностей).

- Таблицы состоят из колонок и строк (кортежей).
- На пересечении каждого столбца и строки находится только одно значение.
- У каждого столбца есть свое имя, которое служит его названием, и все значения в одном столбце имеют один тип.
- Строки в реляционной базе данных неупорядочены. В БД нет "первой", "последней" или 43-й строки, *реляционная бд представляет* собой просто неупорядоченный набор строк.
- Столбцы располагаются в определенном порядке, который задается при создании таблицы. В таблице может не быть ни одной строчки, но обязательно должен быть хотя бы один столбец.
- Запросы к базе данных возвращают результат в виде таблиц, которые тоже могут выступать как объект запросов.

Наиболее распространенные реляционные СУБД: MySQL, MS SQL, Oracle, Postgre SQL, DB2.

Ключ – это столбец (может быть несколько столбцов), добавляемый к таблице и позволяющий установить связь с записями в другой таблице. Существуют **ключи двух типов: первичные и вторичные или внешние.**

Первичный ключ

Под первичным ключом понимают поле или набор полей (составной ключ), однозначно (уникально) идентифицирующих запись. Первичный ключ должен быть минимально достаточным: в нем не должно быть полей, удаление которых из первичного ключа не отразится на его уникальности.

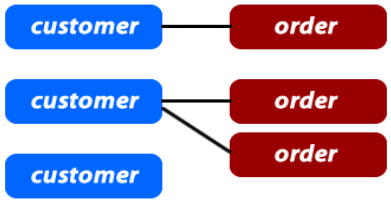
По способу задания первичных ключей различают **логические** (естественные) ключи и **суррогатные** (искусственные).

Внешний (вторичный) ключ - это одно или несколько полей (столбцов) в таблице, содержащих ссылку на поле или поля первичного ключа в другой таблице. Внешний ключ определяет способ объединения таблиц.
Из двух логически связанных таблиц одну называют таблицей первичного ключа или главной таблицей, а другую таблицей вторичного (внешнего) ключа или подчиненной таблицей. СУБД позволяют сопоставить родственные записи из обеих таблиц и совместно вывести их в форме, отчете или запросе.

Между двумя или более таблицами базы данных могут существовать отношения подчиненности. Отношения подчиненности определяют, что для каждой записи главной таблицы *{master,* называемой *еще родительской}* может существовать одна или несколько записей в подчиненной таблице *{detail,* называемой еще *дочерней}.*

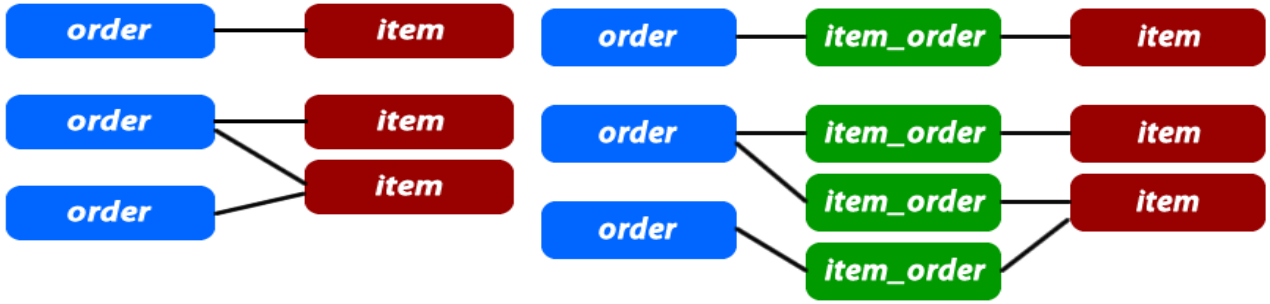
Существует три разновидности связей между таблицами базы данных:

- «один-ко-многим» (когда одной записи родительской таблицы может соответствовать несколько записей в дочерней таблице.). Каждый покупатель может иметь 0 или более заказов. Но каждый заказ может принадлежать только одному покупателю.



- «один-к-одному» (одной записи в родительской таблице соответствует одна запись в дочерней таблице.). Как правило, в БД желательно отсутствие связей 1:1. Данный вид связи «убирается» путем слияния двух таблиц в одну.

- «многие-ко-многим». Пример: каждый заказ может содержать множество товаров. И каждый товар может присутствовать во многих заказах. Данный вид связи заменяется на 2 связи типа “один-ко-многим”



Нормализация БД

Нормализацией схемы базы данных называется процедура, производимая над базой данных с целью удаления в ней избыточности.

Принципы нормализации:

- В каждой таблице БД не должно быть повторяющихся полей;
- В каждой таблице должен быть уникальный идентификатор (первичный ключ);
- Каждому значению первичного ключа должна соответствовать достаточная информация о типе сущности или об объекте таблицы (например, информация об успеваемости, о группе или студентах);
- Изменение значений в полях таблицы не должно влиять на информацию в других полях (кроме изменений в полях ключа).

Пример. Table EMPLOYESS (NAME STRING, JOB_TITLE STRING…)

Эта таблица избыточна — для каждого из сотрудников повторяются одинаковые названия профессий. То есть схема такой базы данных не нормализована. Для небольшой базы данных это не критично, но в больших по объему базах это скажется на размере базы и, в конечном счете, на скорости доступа.

Для нормализации необходимо разбить эту таблицу на две — профессий и фамилий сотрудников.

Table EMPLOYESS (NAME STRING, JOB_ID NUMBER…), JOBS (ID NUMBER, NAME STRING)

Замечание. Хотя в теории баз данных и говорится о том, что схема базы данных должна быть полностью нормализована, в реальности при работе с полностью нормализованными базами данных необходимо применять весьма сложные SQL запросы, что приводит к обратному эффекту — замедлению работы базы данных. Поэтому иногда для упрощения запросов даже прибегают к обратной процедуре — денормализации.

Нормальные формы

Нормальная форма — свойство отношения в реляционной модели данных, характеризующее его с точки зрения избыточности, потенциально приводящей к логически ошибочным результатам выборки или изменения данных. Нормальная форма определяется как совокупность требований, которым должно удовлетворять отношение.

1NF

- Все строки должны быть различными.

- Все элементы внутри ячеек должны быть атомарными (не списками). Другими словами, элемент является атомарным, если его нельзя разделить на части, которые могут использовать в таблице независимо друг от друга.

- **Методы приведения к 1NF:**
  - Устраняем повторяющиеся группы в отдельных таблицах (одинаковые строки).
- Выносим в отдельную таблицу для каждого набора связанных данных.

Код сотрудника	ФИО	Должность	Проекты
1	Иванов	Программист	ID: 123; Название: Система управления паровым котлом; Дата сдачи: 30.09.2011 ID: 231; Название: ПС для контроля и оповещения о превышениях ПДК различных газов в помещении; Дата сдачи: 30.11.2011 ID: 321; Название: Модуль распознавания лиц для защитной системы; Дата сдачи: 01.12.2011

Код сотрудника	ФИО	Должность	Код проекта	Наименование	Дата сдачи
1	Иванов	Программист	123	Система управления паровым котлом	30.09.2011
1	Иванов	Программист	231	ПС для контроля и оповещения о превышениях ПДК различных газов в помещении	30.11.2011
1	Иванов	Программист	321	Название: Модуль распознавания лиц для защитной системы	01.12.2011

2NF

Таблица находится во 2NF если она находится в 1NF и столбцы, не входящие в полный первичный ключ, должны зависеть от полного первичного ключа

Примечание. Таблицы с простыми ключами , находящиеся в 1NF всегда находятся во 2NF.

Методы приведения к 2NF: выносим часть ключа и зависимые атрибуты в отдельные таблицы.

В таблице первичным ключем объявлены поля archSurname (фамилия архитектора) и title (название станции). Если таблица следует нормальной форме, то все поля, не входящие в первичный ключ, зависят от него. Итак, смотрим: archName (имя архитектора) зависит от archSurname, и это логично. Поле completedDate (дата сдачи станции в эксплуатацию) зависит от title (от ее названия), это тоже логично.

archName :varchar(255)	archSurname: varcahr(255)-PK	title: varchar(255)-PK	completedDate :date
Александр	Андреев	Площадь Ленина	1958.06.01
Александр	Жук	Достоевская	1991.12.30

Таблица Stations.

Но имя архитектора совершенно не зависит от названия станции, а дата сдачи станции, в свою очередь, никак не зависит от имени архитектора. Следовательно, два поля таблицы не зависят от соответственных полей в первичном ключе и таблица не соответствует второй нормальной форме.

Раскидываем данные по двум таблицам: Stations и Architectors. При этом нам понадобится третья, связующая таблица (что, плюс ко всему, позволит присваивать один проект нескольким архитекторам). В результате у нас получаются три таблицы:

id: int-PK	title completed	Date :date
1	Площадь Ленина	1958.06.01
2	Достоевская	1991.12.30

Таблица Stations.

id :int-PK	name :varchar(255)	surname: varcahr(255)-PK
1	Александр	Жук
2	Александр	Андреев

Таблица Architectors.

stationId :int-FK1	archId :int-FK2
1	2
2	1

Таблица StationArchitectors.

3NF

Таблица находится во 3NF если она находится в 2NF и все столбцы, не входящие в полный первичный ключ, должны зависеть от него и не должны зависеть друг от друга.

Методы приведения к 3NF: выносим зависимые атрибуты в отдельную таблицу.

В этой таблице ключом является MemberId. Ясно что City, Street, State зависят от ZipCode.

Имя поля	*Тип данных*
MemberId	integer
FirstName	nvarchar(50)
LastName	nvarchar(50)
DateOfBirth	date
Street	varchar(100)
City	varchar(75)
State	varchar(75)
ZipCode	varchar(12)
Email	varchar(200)
DateOfJoining	date

Выносим поля City, Street, State в отдельную таблицу у которой PK=ZipCode и устанавливаем FK на новую таблицу.

Имя поля	*Тип данных*
ZipCode	varchar(12)
Street	varchar(100)
City	varchar(75)
State	varchar(75)

Имя поля	*Тип данных*
MemberId	integer
FirstName	nvarchar(50)
LastName	nvarchar(50)
DateOfBirth	date
ZipCode	varchar(12)
Email	varchar(200)
DateOfJoining	date

Транзакции

В SQL:1999 поддерживается классическое понимание транзакции, характеризуемое аббревиатурой ACID (от Atomicy, Consistency, Isolation и Durability). В соответствии с этим понятием под транзакцией разумеется последовательность операций (например, над базой данных), обладающая следующими свойствами.

- *Атомарность* (Atomicy). Это свойство означает, что результаты всех операций, успешно выполненных в пределах транзакции, должны быть отражены в состоянии базы данных, либо в состоянии базы данных не должно быть отражено действие ни одной операции (конечно, здесь речь идет об операциях, изменяющих состояние базы данных). Свойство атомарности, которое часто называют свойством «все или ничего», позволяет относиться к транзакции, как к динамически образуемой составной операции над базой данных.

- *Согласованность* (Consistency). В классическом смысле это свойство означает, что транзакция может быть успешно завершена с *фиксацией* результатов своих операций только в том случае, когда действия операций не нарушают целостность базы данных, т. е. удовлетворяют набору ограничений целостности, определенных для этой базы данных. В стандарте SQL:1999 это свойство расширяется тем, что во время выполнения транзакции разрешается устанавливать точки согласованности (см. ниже про точки сохранения) и явным образом проверять ограничения целостности.

- *Изоляция* (Isolation). Требуется, чтобы две одновременно выполняемые транзакции никоим образом не действовали одна на другую. Другими словами, результаты выполнения операций транзакции T1 не должны быть видны никакой другой транзакции T2 до тех пор, пока транзакция T1 не завершится успешным образом.

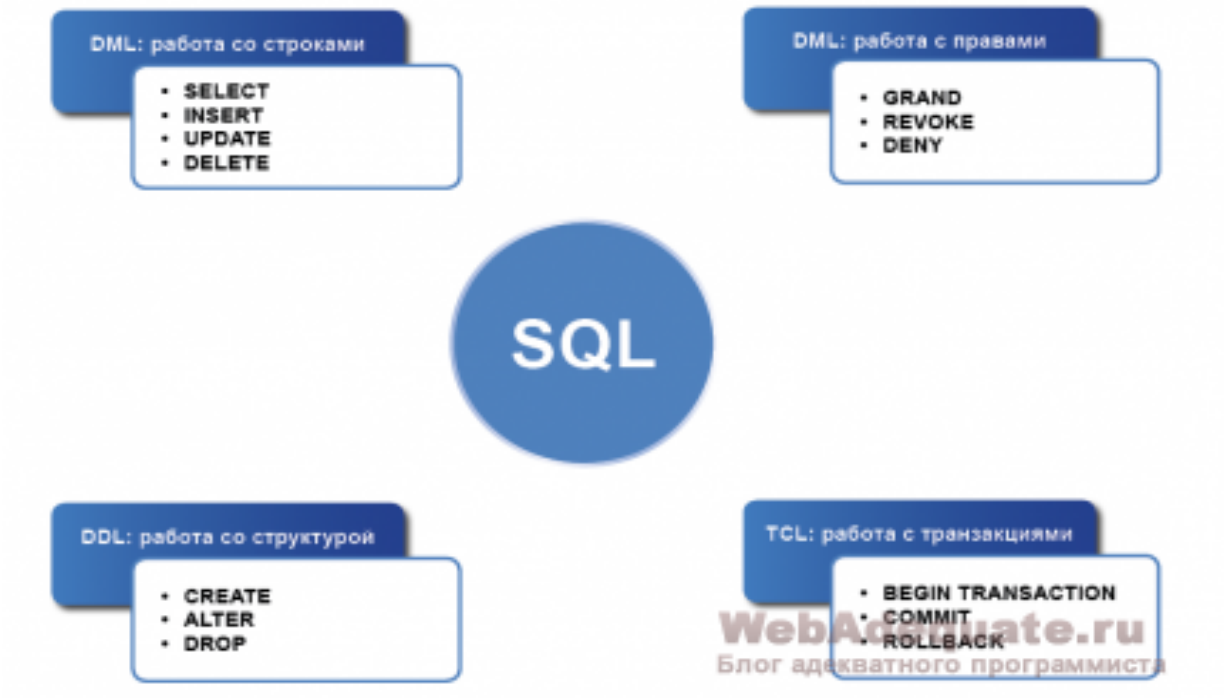
- *Долговечность* (Durability). После успешного завершения транзакции все изменения, которые были внесены в состояние базы данных операциями этой транзакции, должны гарантированно сохраняться, даже в случае сбоев аппаратуры или программного обеспечения.

У каждой выполняемой транзакции имеются три характеристики, значения которых существенно влияют на действия системы при управлении транзакцией, – уровень изоляции (isolation level), режим доступа (access mode) и размер области диагностики. При неявном образовании транзакции эти характеристики устанавливаются по умолчанию: транзакция получает максимальный уровень изоляции от одновременно выполняемых транзакций; режим доступа, позволяющий выполнять и операции выборки, и операции обновления базы данных; и назначаемый по умолчанию размер области диагностики.

isolation_level ::= READ UNCOMMITED | READ COMITTED | REPEATABLE READ | SERIALIZABLE

access_mode ::= READ ONLY | READ WRITE

Классификация SQL команд



Data Manipulation Language (работа со строками)

- INSERT — добавление строк(и);
- SELECT — выборка строк(и);
- UPDATE — изменение строк(и);
- DELETE— удаление строк(и);

Data Definition Language (работа со структурой базы)

- CREATE — создание объекта (например таблицы);
- ALTER— изменение объекта (например добавление/изменение полей);
- DROP — удаление объекта;

Data Control Language (работа с правами доступа)

- GRANT — назначения прав пользователю;
- DENY — явный запрет для пользователя;
- REVOKE — отменить запрет или разрешение;

Transaction Control Language (работа с транзакциями)

- BEGIN TRANSACTION — начать транзакцию;
- COMMIT — принять изменения внесенные текущей транзакцией;
- ROLLBACK — откат;

Команды DML

SELECT * | { [DISTINCT | ALL] <value expression>.,..}

FROM { <table name> [<alias>] }.,..

[WHERE <predicate>]

[GROUP BY { <column name> | <integer> }.,..]

[HAVING <predicate>]

[ORDERBY { <column name> | <integer> }.,..]

[{ UNION [ALL]

SELECT * | { [DISTINCT | ALL] < value expression >.,..}

FROM { <table name> [<alias>]} .,..

[WHERE <predicate>

[GROUP BY { <columnname> | <integer> }.,..]

[HAVING <predicate>]

[ORDER BY { <columnname> | <integer> },..] }] ...;

UPDATE

UPDATE <tablename>

SET { | }.,. .< column name> = <value expresslon> [WHERE <predlcate>

| WHERE CURRENT OF <cursor name> (*только для вложения*)];

INSERT

INSERT INTO < table name> [(<column name> .,.]

{ VALUES (<value expression> .,..) } | <query>;

DELETE

DELETE FROM <table name>

[WHERE <predicate>

| WHERE CURRENT OF <cursor name> (*только для вложения*)];

Движки MySQL

MySQL предоставляет широкий выбор вариантов хранения информации на сервере. Эти механизмы хранения называются **движками** или **Storage Engine**. У каждого движка есть свои неоспоримые плюсы и минусы, на основании которых уже можно и нужно сделать вывод, на каком из них хранить информацию базы данных. У всех этих движков свои разработчики, авторы и появлялись в сборке сервера постепенно по мере его развития. Более подробную информацию можно найти на сайте разработчиков **mysql.com** в разделе документации.

InnoDB. Один их самых популярных используемых движков. Поддерживает почти весь функционал сервера MySQL. Используется по умолчанию. Данных движок хорошо себя показывает на больших проектах с высокой нагрузкой. Лимит дискового пространства **64 Тб**.

Особенности:

- Хранит все данных в одном файле.

Плюсы:

- Поддержка транзакций.
- Поддержка внешних ключей.
- Позволяет управлять конкурентным доступом с помощью многоверсионности (**MVCC**).
- Поддержка блокировок таблиц (LOCK).
- Быстрая выборка (SELECT).
- Индексация полей.

Минусы:

- Не поддерживает полнотекстовый поиск.

MyISAM. Очень популярный движок для web-проектов. Очень быстрая вставка и изменение записей, за счет отсутствия внешних ключей и транзакций. В отличии от InnoDB, все данные хранит по отдельности (все базы данных лежат в папке C:/SERVER/**MySQL/Data/**, 1 папка - 1 база): файлы с расширением **.MYI** (MYIndex) содержат индексы полей. Файлы с расширением **.MYD** (MYData) содержат сами данные. Файлы с расширением **.frm** (format) содержат структуру таблиц. Чаще всего используется для справочников, Лимит дискового пространства **256 Тб**.

Особенности:

- Отсутствует поддержка транзакций.
- Не поддерживает внешние ключи.
- Хранит все данные отдельно друг от друга.

Плюсы:

- Поддержка блокировок таблиц (LOCK).
- Быстрая вставка строк (INSERT).
- Быстрое изменение строк (UPDATE).
- Быстрое удаление строк (DELETE).
- Поддержка полнотекстового поиска.
- Индексация полей.

Минусы:

- Гораздо **медленнее** производит выборку (SELECT), чем **InnoDB**, да и многие другие.
- Отсутствует возможность восстановления данных по журналу.
- Отсутствуют инструменты резервного копирования.
- Слабая реализация сортировки.
- Риски нарушения стабилизации работы, потери данных.

При создании БД под Linux по умолчанию используется движок MyISAM, для Windows – InnoDB.

Литература

1. SQL полное руководство. Джеймс Р. Грофф, Пол Н. Вайнберг
2. Понимание SQL. Мартин Грубер