Введение в понятие «Целостность данных»

Лекция 5

Понятие целостности (Wiki...

Це́лостность ба́зы да́нных (англ. database integrity) — соответствие имеющейся в базе данных информации её внутренней логике, структуре и всем явно заданным правилам.

Каждое правило, налагающее некоторое ограничение на возможное состояние базы данных, ***называется ограничением целостности***.

Примеры правил: вес детали должен быть положительным; количество знаков в телефонном номере не должно превышать 15; возраст родителей не может быть меньше возраста их биологического ребёнка и так далее.

Понятие целостности (Wiki…

В теории реляционных баз данных принято выделять четыре типа ограничений целостности:

1. *Ограничением базы данных* называется ограничение на значения, которые разрешено принимать указанной базе данных.
2. *Ограничением переменной отношения* называется ограничение на значения, которые разрешено принимать указанной переменной отношения.
3. *Ограничением атрибута* называется ограничение на значения, которые разрешено принимать указанному атрибуту.
4. *Ограничение типа* представляет собой ни что иное, как определение множества значений, из которых состоит данный тип.

Примером распространённого ограничения уровня переменной отношения является потенциальный ключ (подмножество атрибутов отношения, удовлетворяющее требованиям *уникальности* и *несократимости* (*минимальности*); примером распространённого ограничения уровня базы данных является внешний ключ.

Типы ограничений целостности данных:

- обязательные данные;
- ограничения для доменов полей;
- целостность сущностей;
- ссылочная целостность.
- корпоративные ограничения (*базы данных)*;

Типы ограничений целостности данных

Обязательные данные

Некоторые поля всегда должны содержать одно из допустимых значений, другими словами, эти поля не могут иметь пустого значения.

Ограничения для доменов полей (Домен — тип данных, то есть множество допустимых значений)

Каждое поле имеет свой домен, представляющий собой набор его допустимых значений.

Корпоративные ограничения целостности

Существует понятие "корпоративные ограничения целостности" как дополнительные правила поддержки целостности данных, определяемые пользователями, принятые на предприятии или администраторами баз данных. Ограничения предприятия называются бизнес-правилами.

Целостность сущностей

- Это ограничение целостности касается **первичных ключей** базовых таблиц. По определению, **первичный ключ** – минимальный идентификатор (одно или несколько полей), который используется для уникальной идентификации записей в таблице. Таким образом, никакое подмножество **первичного ключа** не может быть достаточным для уникальной идентификации записей.
- **Целостность сущностей** определяет, что в базовой таблице ни одно поле **первичного ключа** не может содержать отсутствующих значений, обозначенных **NULL**.
- Если допустить присутствие определителя **NULL** в любой части **первичного ключа**, это равносильно утверждению, что не все его поля необходимы для уникальной идентификации записей, и противоречит определению **первичного ключа**.

Ссылочная целостность

Указанное ограничение целостности касается внешних ключей. **Внешний ключ** – это поле (или множество полей) одной таблицы, являющееся ключом другой (или той же самой) таблицы. Внешние ключи используются для установления логических связей между таблицами. Связь устанавливается путем присвоения значений внешнего ключа одной таблицы значениям ключа другой.

Между двумя или более таблицами базы данных могут существовать отношения подчиненности, которые определяют, что для каждой записи **главной таблицы** (называемой еще **родительской**) может существовать одна или несколько записей в **подчиненной таблице** (называемой также **дочерней**).

Существует три разновидности связи между таблицами базы данных:

"один-ко-многим";

"один-к-одному";

"многие-ко-многим".

Ссылочная целостность

- Часто связь между таблицами устанавливается по **первичному ключу**, т.е. значениям **внешнего ключа** одной таблицы присваиваются значения **первичного ключа** другой. Однако это не является обязательным – в общем случае связь может устанавливаться и с помощью вторичных ключей.
- Кроме того, при установлении связей между таблицами не требуется непременная уникальность ключа, обеспечивающего установление связи.
- Поля внешнего ключа не обязаны иметь те же имена, что и имена ключей, которым они соответствуют.
- Внешний ключ может ссылаться на свою собственную таблицу – в таком случае внешний ключ называется ***рекурсивным***.

- Ссылочная целостность определяет: если в таблице существует **внешний ключ**, то его значение должно либо соответствовать значению **первичного ключа** некоторой записи в базовой таблице, либо задаваться определителем **NULL**.

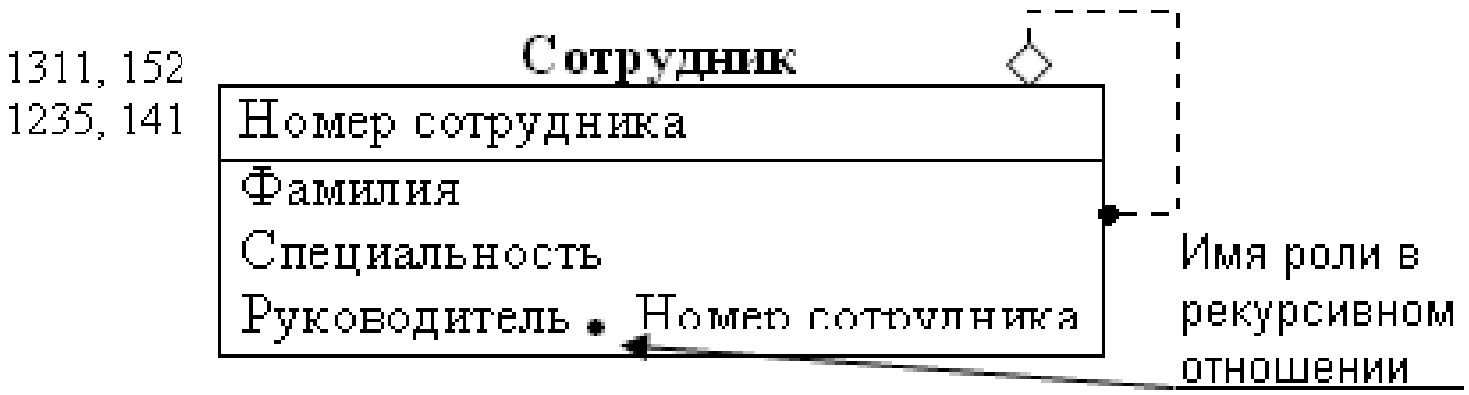
Ссылочная целостность

Внешний ключ может ссылаться на свою собственную таблицу – в таком случае внешний ключ называется ***рекурсивным***.

Рассмотрим таблицу **СОТРУДНИК**, в которой содержатся сведения о сотрудниках и их руководителях. Например, сотрудник с номером 1311 является руководителем сотрудника 1235. Сотрудники Сидоров и Чернов не имеют руководителей

Сотрудник

Номер сотрудника	Фамилия	Специальность	Руководитель
1235	Иванов	Электрик	1311
1412	Петров	Штукатур	1520
1311	Сидоров	Электрик	
1520	Чернов	Штукатур	



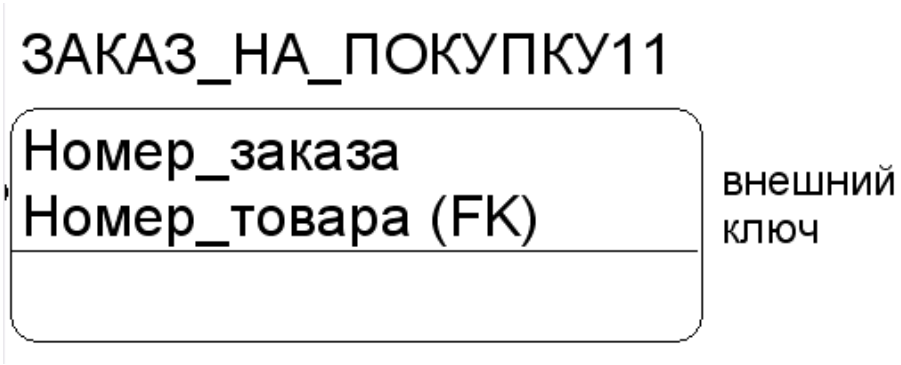
Несколько важных моментов, связанных с внешними ключами

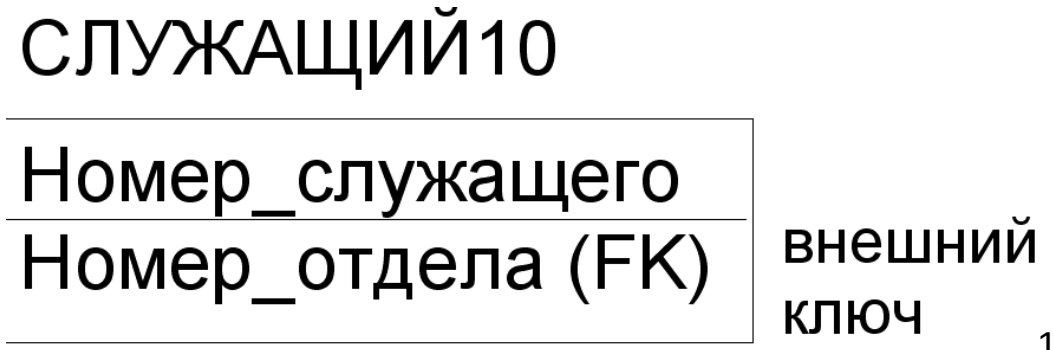
- Следует проанализировать, допустимо ли использование во внешних ключах пустых значений.
- В общем случае, если участие дочерней таблицы в связи является обязательным, то рекомендуется запрещать применение **пустых значений** в соответствующем **внешнем ключе**.
- В то же время, если имеет место частичное участие дочерней таблицы в связи, то помещение пустых значений в поле внешнего ключа должно быть разрешено.
  - Например, если в операции фиксации сделок некоторой торговой фирмы необходимо указать покупателя, то поле **КодКлиента** должно иметь атрибут **NOT NULL**. Если допускается продажа или покупка товара без указания клиента, то для поля **КодКлиента** допустим атрибут **NULL**.

Несколько важных моментов, связанных с внешними ключами

Если все атрибуты первичного ключа сущности-родителя наследуются в качестве части первичного ключа сущности-потомка, то отношение называется **идентифицирующим**.

Если какой-нибудь из наследуемых атрибутов не является частью первичного ключа, то отношение называется **неидентифицирующим**.





Организация поддержки ссылочной целостности

При выполнении операций модификации данных в базе возможны следующие ситуации:

1. Вставка новой строки в дочернюю таблицу. Для обеспечения ссылочной целостности необходимо убедиться, что значение внешнего ключа новой строки дочерней таблицы равно пустому значению либо некоторому конкретному значению, присутствующему в поле первичного ключа одной из строк родительской таблицы.
2. Удаление строки из дочерней таблицы. Никаких нарушений ссылочной целостности не происходит.
3. Обновление внешнего ключа в строке дочерней таблицы. Этот случай подобен описанной выше первой ситуации. Для сохранения ссылочной целостности необходимо убедиться, что значение внешнего ключа в обновленной строке дочерней таблицы равно пустому значению либо некоторому конкретному значению, присутствующему в поле первичного ключа одной из строк родительской таблицы.
4. Вставка строки в родительскую таблицу. Такая вставка не может вызвать нарушения ссылочной целостности. Добавленная строка просто становится родительским объектом, не имеющим дочерних объектов.

Организация поддержки ссылочной целостности

Удаление строки из родительской таблицы. Ссылочная целостность окажется нарушенной, если в дочерней таблице будут существовать строки, ссылающиеся на удаленную строку родительской таблицы. В этом случае может использоваться одна из следующих стратегий:

NO ACTION. Удаление строки из родительской таблицы запрещается, если в дочерней таблице существует хотя бы одна ссылающаяся на нее строка.

CASCADE. При удалении строки из родительской таблицы автоматически удаляются все ссылающиеся на нее строки дочерней таблицы (удаление строки родительской таблицы автоматически распространяется на любые дочерние таблицы).

SET NULL. При удалении строки из родительской таблицы во всех ссылающихся на нее строках дочернего отношения в поле внешнего ключа, соответствующего первичному ключу удаленной строки, записывается пустое значение. Эта стратегия может использоваться, только когда в поле внешнего ключа дочерней таблицы разрешается помещать пустые значения.

SET DEFAULT. При удалении строки из родительской таблицы в поле внешнего ключа всех ссылающихся на нее строк дочерней таблицы автоматически помещается значение, указанное для этого поля как значение по умолчанию. Эта стратегия применима лишь в тех случаях, когда полю внешнего ключа дочерней таблицы назначено некоторое значение, принимаемое по умолчанию.

NO CHECK. При удалении строки из родительской таблицы никаких действий по сохранению ссылочной целостности данных не предпринимается.

Обновление первичного ключа в строке родительской таблицы:

Если значение первичного ключа некоторой строки родительской таблицы будет обновлено, нарушение ссылочной целостности случится при том условии, что в дочернем отношении существуют строки, ссылающиеся на исходное значение первичного ключа.

Для сохранения ссылочной целостности может применяться любая из описанных выше стратегий. При использовании стратегии **CASCADE** обновление значения первичного ключа в строке родительской таблицы будет отображено в любой строке дочерней таблицы, ссылающейся на данную строку.

Семантическая целостность базы данных

- Существует и другой вид целостности – смысловая (семантическая) целостность базы данных.
- Требование ***смысловой целостности*** определяет, что данные в базе данных должны изменяться таким образом, чтобы не нарушалась сложившаяся между ними смысловая связь.
- Уровень поддержания целостности данных в разных системах существенно варьируется.

Идеология архитектуры клиент-сервер

требует переноса максимально возможного числа правил целостности данных на сервер.

- К преимуществам такого подхода относятся:
  - гарантия целостности базы данных, поскольку все правила сосредоточены в одном месте (в базе данных);
  - автоматическое применение определенных на сервере ограничений целостности для любых приложений;
  - отсутствие различных реализаций ограничений в разных клиентских приложениях, работающих с базой данных;
  - быстрое срабатывание ограничений, поскольку они реализованы на сервере и, следовательно, нет необходимости посылать данные клиенту, увеличивая при этом сетевой трафик;
  - доступность внесенных в ограничения на сервере изменений для всех клиентских приложений, работающих с базой данных, и отсутствие необходимости повторного распространения измененных приложений клиентов среди пользователей.

Идеология архитектуры клиент-сервер

К недостаткам хранения ограничений целостности на сервере можно отнести:

отсутствие у клиентского приложения возможности реагировать на некоторые ошибочные ситуации, возникающие на сервере при реализации тех или иных правил (например, ошибок при выполнении хранимых процедур на сервере);

ограниченность возможностей языка SQL и языка хранимых процедур и триггеров для реализации всех возникающих потребностей определения целостности данных.

На практике в клиентских приложениях реализуют лишь такие правила, которые тяжело или невозможно реализовать с применением средств сервера.

Все остальные ограничения целостности данных переносятся на сервер.

Таблицы с ограничениями в стандарте языка SQL

Рассмотрим предусмотренные стандартом языка функции, которые предназначены для поддержания целостности данных. Эта поддержка включает средства задания ограничений, они вводятся с целью защиты базы данных от нарушения согласованности сохраняемых в ней данных.

К таким типам поддержки целостности данных относятся:

- обязательные данные;
- ограничения для доменов полей;
- целостность сущностей;
- ссылочная целостность;
- требования конкретного предприятия.

Большая часть перечисленных ограничений задается в операторах CREATE TABLE и ALTER TABLE.

Создание таблицы

CREATE TABLE имя_таблицы

{(имя_столбца тип_данных [NOT NULL][UNIQUE][DEFAULT <значение>]

[CHECK (<условие_выбора>)][,...n]}

[CONSTRAINT имя_ограничения]

[PRIMARY KEY (имя_столбца [,...n])

{[UNIQUE (имя_столбца [,...n])}

[FOREIGN KEY (имя_столбца_внешнего_ключа [,...n])

REFERENCES имя_род_таблицы[(имя_столбца_род_таблицы [,...n])],

[MATCH {PARTIAL | FULL}]

[ON UPDATE {CASCADE|SET NULL|SET DEFAULT|NO ACTION}]

[ON DELETE {CASCADE| SET NULL |SET DEFAULT|NO ACTION}]{[CHECK(<условие_выбора>)][,...n]})

Изменение и удаление таблицы

Для внесения изменений в уже созданные таблицы стандартом SQL предусмотрен оператор **ALTER TABLE**, предназначенный для выполнения следующих действий:

добавление в таблицу нового столбца;

удаление столбца из таблицы;

добавление в определение таблицы нового ограничения;

удаление из определения таблицы существующего ограничения;

задание для столбца значения по умолчанию;

отмена для столбца значения по умолчанию.

Обобщенный формат оператора ALTER TABLE

ALTER TABLE имя_таблицы

[ADD [COLUMN]имя_столбца тип_данных [NOT NULL][UNIQUE]

[DEFAULT <значение>][CHECK (<условие_выбора>)]]

[DROP [COLUMN] имя_столбца [RESTRICT | CASCADE]]

[ADD [CONSTRAINT [имя_ограничения]]

[{PRIMARY KEY (имя_столбца [,...n]) |[UNIQUE (имя_столбца [,...n])}

|[FOREIGN KEY (имя_столбца_внешнего_ключа [,...n])
REFERENCES имя_род_таблицы
[(имя_столбца_род_таблицы [,...n])],

[MATCH {PARTIAL | FULL} [ON UPDATE {CASCADE| SET NULL |
SET DEFAULT | NO ACTION}] [ON DELETE {CASCADE| SET
NULL | SET DEFAULT | NO ACTION}]
|[CHECK(<условие_выбора>)][,...n]}]

[DROP CONSTRAINT имя_ограничения [RESTRICT | CASCADE]]

[ALTER [COLUMN] SET DEFAULT <значение>]

[ALTER [COLUMN] DROP DEFAULT]

Представления

(VIEW), представляют собой временные, производные (виртуальные) таблицы и являются объектами базы данных, информация в которых не хранится постоянно, как в базовых таблицах, а формируется динамически при обращении к ним.

Обычные таблицы относятся к базовым, т.е. содержащим данные и постоянно находящимся на устройстве хранения информации. Представление не может существовать само по себе, а определяется только в терминах одной или нескольких таблиц.

Применение представлений позволяет разработчику базы данных обеспечить каждому пользователю или группе пользователей наиболее подходящие способы работы с данными, что решает проблему простоты их использования и безопасности.

Содержимое представлений выбирается из других таблиц с помощью выполнения запроса, причем при изменении значений в таблицах данные в представлении автоматически меняются.

Представление - это фактически тот же запрос, который выполняется всякий раз при участии в какой-либо команде.

Результат выполнения этого запроса в каждый момент времени становится содержанием представления.

У пользователя создается впечатление, что он работает с настоящей, реально существующей таблицей.

Представления

У СУБД есть две возможности реализации представлений.

Если его определение простое, то система формирует каждую запись представления по мере необходимости, постепенно считывая исходные данные из базовых таблиц.

В случае сложного определения СУБД приходится сначала выполнить такую операцию, как материализация представления, (сохранить информацию, из которой состоит представление, во временной таблице). Затем система приступает к выполнению пользовательской команды и формированию ее результатов, после чего временная таблица удаляется.

Представление - это предопределенный запрос, хранящийся в базе данных, который выглядит подобно обычной таблице и не требует для своего хранения дисковой памяти.

Для хранения представления используется только оперативная память.

В отличие от других объектов базы данных представление не занимает дисковой памяти за исключением памяти, необходимой для хранения определения самого представления.

Представление может содержать:

- подмножество записей из таблицы БД, отвечающее определённым условиям (например, при наличии одной таблицы «Люди» можно создать два представления «Мужчины» и «Женщины», в каждом из которых будут записи только о людях соответствующего пола);
- подмножество столбцов таблицы БД, требуемое программой (например, из реальной таблицы «Сотрудники» представление может содержать по каждому сотруднику только ФИО и табельный номер);
- результат обработки данных таблицы определёнными операциями (например, представление может содержать все данные реальной таблицы, но с приведением строк в верхний регистр и обрезанными начальными и концевыми пробелами);
- результат объединения (join) нескольких таблиц (например, при наличии таблиц «Люди», «Адреса», «Улицы», «Фирмы и организации» возможно построение представления, которое будет выглядеть как таблица, для каждого человека содержащее его личные данные, адрес места жительства, название организации, где он работает, и адрес этой организации);
- результат слияния нескольких таблиц с одинаковыми именами и типами полей, когда в представлении попадают все записи каждой из сливаемых таблиц (возможно, с исключением дублирования);
- результат группировки записей в таблице
- практически любую комбинацию вышеперечисленных возможностей.

Оператор определения представления

CREATE VIEW < ИМЯ ПРЕДСТАВЛЕНИЯ>

[(< СПИСОК СТОЛБЦОВ>)] AS <SQL - ЗАПРОС >

- Если список имен столбцов в представлении не задан, то каждый столбец представления получает имя соответствующего столбца запроса.

SQL CREATE VIEW

- **Пример 1.**Простое представление, которое создается из данных одной таблицы:

CREATE VIEW London_view **AS SELECT** * **FROM** Salespeople **WHERE** city = ‘London’;

- **Пример 2.**При создании представления можно можно задать новые имена полей:

CREATE VIEW Rating_view(rating,number) **AS SELECT** rating, **COUNT**(*) **FROM** Customers **GROUP BY** rating;

Пример 3.Представления могут получать информацию из любого количества базовых таблиц:

CREATE VIEW Nameorders **AS SELECT** onum, amt,a.snum, sname, cname **FROM** Orders a, Customers b, Salespeople C **WHERE** a.cnum = b.cnum **AND** a.snum = c.snum;

- **Пример 4**.При создании представлений можно использовать подзапросы, включая и связанные подзапросы:

CREATE VIEW Sales_view *AS SELECT* b.odate, a.snum, a.sname, *FROM* Salespeople a, Orders b *WHERE* a.snum = b.snum *AND* b.amt = (*SELECT MAX*(amt) *FROM*Orders c *WHERE* c.odate = b.odate);

- **Пример 5.**

CREATE VIEW empl_v04 *AS SELECT* e.eid, e.sname, e.fname, e.otch, p.pname, d.dname *FROM* posts p, departments d, employees e *WHERE* e.did = d.did *AND* e.pid = p.pid;