Учим SQL. Алан Бьюли.
Выражения, с помощью которых создаются объекты БД(таблицы, индексы, ограничения) - это SQL-выражения управления схемой данных(schema statments)

Выражения, предназначенные для создания, манипулирования и извлечения данных - data statments

ER-диаграмма - entity-relationship diagram - диаграмма сущностей и связей

В реляционной модели БД используются избыточные данный для связи между таблицами

Столбцов обычно много можно делать. Например, Microsoft SQL сервер позволяет создать до 1024 слолбцов. Число строк в таблицах, как правило, - вопрос физических возможностей системы.

Первичный ключ, составленный из двух и более столбцов называется составным (compound key).

Внешним ключом (foreign key) называется столбец ссылающийся на столбец с уникальными значениями, идектифицирующими определенные записи

Нормализация - процесс улучшения стуктуры базы данных, с целью обеспечения хранения всех независимых элементов данных только в одном месте (за исключением внешних ключей)

Строка (row) и запись (record) это одно и то же

Внешний ключ может состоять из нескольких столбцов

В SQL имеются возможности для обеспечение объектно ориентированной функциональности

В реляционной модели БД можно создать новую таблицу с данными и полями результирующего набора

Все элементы БД, созданные посредством SQL-выражений для управления схемой, хранятся в специальный таблицах, называемых словарем данных (data dictionary). Все эти данные о базе данных называются метаданными.
К таблицам словаря данных можно делать select-запросы.

Способ выполнения SQL-выражения зависит от компогнента механизма СУБД (database engine), называемого оптимизатором (optimizer).
Оптимизатор рассматривает SQL-выражение и с учетом конфигурации

таблиц и доступных индексов и принимает решение о самом эффективном пути выполнения запроса.
Большинство СУБД позволяют влиять на работу оптимизатора при помощи т.н. покдсказок (optimizer hints).

Как правило, для работы с данными требуется определенная логика, которую не может предоставить SQL. Поэтому приходится интегрировать свою БД с каких-нибудь языком программирования.
Некоторые производители БД интегрируют свои системы сосвоими языками, например, Oracle с PL/SQL или Microsoft с TransactSQL

После выполнения SQL-выражения, СУБД должна вернуть количество строк, которые были затронуты. Это касается всех типов работы с данными (select, insert, update, delete)

Некоторые СУБД не выполняют запросы без ключевого слова from.
Например, select now() не будет работать в Oracle.
Для подобного рода запросов имеется специальная таблица dual. Для совместимости MySQL тоже предоставляет эту таблицу.
SELECT now() FROM dual; выведет текущую дату.

Символьные данные.
Символьные данные могут храниться как строки фиксированной (char), так и переменной (varchar) длины. Разница в том, что строки фиксированной длины справа дополняются пробелами, а строки переменной - нет.
макс длинна:
char - 255
varchar - 255 до версии 5.0.3, и 65535, начиная с версии 5.0.3

В Oracle для столбцов char - 2000, а для varchar - 4000 байтов масксимальная длина

В Oracle для столбцов со строками переменной длины используется тип varchar2

Текстовые данные.
Текстовые данные используются, если необходимо хранить количество символов, больше чем char или varchar
tinytext - 255 (2^8 – 1)
text - 65535 (2^16 – 1)
MEDIUMTEXT - 16777215 (2^24 – 1)
LONGTEXT - 4294967295 (2^32 – 1)
blob - 65535 (2^16 – 1)
MEDIUMBLOB - 16777215 (2^24 – 1)
LONGBLOB - 4294967295 (2^32 – 1)

Если размер данных превышает размер столбца, то лишнее отсекается.

При добавлении данных в текстовые форматы, пробелы в конце не обрезаются, в отличие от varchar
ПРи использовании столбцов типа text для сортировки и группировки используются только первые 1024 байта. Данный параметр где-то можно изменять, если надо.

В Oracle, для символьных данных большого размера применяется тип clob.

Числовые типы данных.
Все numeric-поля могут быть как со знаком минус, так и беззнаковыми.

Типы данных для целых:
Bit(M), где M от 1 до 64, если M не указана, то по умолчанию равна 1
Tinyint от -128 до 127 или от 0 до 255
Smallint от -32 768 до 32 767 или от 0 до 65 535
Mediumint от -8 388 608 до 8 388 607 или от 0 до 16 777 215
Int от -2 147 483 648 до 2 147 483 647 или от 0 до 4 294 967 295
Bigint от -9 223 372 036 854 775 808 до 9 223 372 036 854 775 807 или от 0 до 18 446 744 073 709 551 615

Типы данных для чисел с плавающей точкой:
Float(M,D) от -3.402823466E+38 до -1.175494351E-38, 0, и от 1.175494351E-38 до 3.402823466E+38.
Double(M,D) от -1.7976931348623157E+308 до -2.2250738585072014E-308, 0, и от 2.2250738585072014E-308 до 1.7976931348623157E+308

где M - общее количество знаков, D - количество знаков после запятой

ВременнЫе типы данных.

Date	YYYY MM DD	от 1000-01-01 до 9999-12-31
Datetime	YYYY MM DD HH:MI:SS	от 1000-01-01 00:00:00 до 9999-12-31 23:59:59
Timestamp	YYYY MM DD HH:MI:SS	от 1970-01-01 00:00:00 до 2037-12-31 23:59:59
Year	YYYY	от 1901 до 2155
Time	HHH:MI:SS	от 838:59:59 до 838:59:59

В MySQL при изменении любого значения строки обновляется поле Timestamp, текущим временем и датой

При создании таблицы необходимо указывать первычеый ключ. Информация об этом указываетсся в конце DDL-ки при помощи ограничения (constraint) для таблицы. Ограничению надо давать имя. Пример: CONSTRAINT pk_person PRIMARY KEY (person_id). Такое ограничение называется ограничением первичного ключа(primary key constraint).

Есть еще проверочное ограничение(check constraint). Оно применяется для ограничения допустимых значений. При определении столбца пишется, например, gender CHAR(1) CHECK (gender IN ('M','F')).
MySQL допускает создание проверочных ограничений, но не выполняет их проверку. Поэтому там есть enum

Пример создания первичного ключа из двух полей: CONSTRAINT pk_favorite_food PRIMARY KEY (person_id, food)

Есть еще ограничение внешнего ключа(foreign key constraint). Для него надо указать имя внешнего ключа и имя таблицы и ключа, на которую внешний ключ ссылается. CONSTRAINT fk_person_id FOREIGN KEY (person_id) REFERENCES person (person_id)

DESC favorite_food - выведет информацию о таблице;

Добавит к таблице автоинкремент: ALTER TABLE person MODIFY person_id SMALLINT UNSIGNED AUTO_INCREMENT;

Вот так выглядит инсерт:
```
> INSERT INTO person (person_id, fname, lname, gender, birth_date)
> VALUES (null, 'William','Turner', 'M', '1972 05 27');
```

Так выглядит апдейт:
```
> UPDATE person
> SET address = '1225 Tremont St.',
> city = 'Boston',
> state = 'MA',
> country = 'USA',
> postal_code = '02138'
> WHERE person_id = 1;
```

А так выглядит делит:
```
> DELETE FROM person
> WHERE person_id = 2;
```

Если мы вставляем данные, где внешнему ключу нет соответствующего первичного ключа в другой таблице, то будет вызвана ошибка. Данные сперка должны вставляться в родительскую таблицу, а затем в дочернюю.

Ограничения внешнего ключа выполняются только если таблица использует механизм хранения InnoDB.

Если в MySQL делать апдейт, где, например, одно поле не будет соответствовать допустимым значениям, то апдейт все равно будет выполнен, но с варнингом.
Варнинги можно смотреть следующим образом:
SHOW WARNINGS;

К примеру, enum будет заполнен пустой строкой '', а поле типа date '0000 00 00'

Для просмотра доступных таблиц используется команда SHOW TABLES;

Удаляются таблицы следующим образом: DROP TABLE favorite_food;

Что бы посмотреть столбцы таблицы, используется команда DESC customer;

Каждому соединению к БД присваивается идентификатор.
При каждом запросе сервер проверяет следующее:
- есть ли у данного пользователя раздешение на выполнение подобного запроса
- есть ли разрешение на доступ к этим данным
- правилен ли синтаксис выражения

После выражение передается оптимизатору запроса, который который определяет наиболее эффективный способ его выполнения. Оптимизатор рассмотрит порядок соединения таблиц, перечисленных в запросе, и доступные индексы, а затем определит план выполнения, используемый сервером при выполнении этого запроса

После выполнения запроса, сервер возвращает в вызывающее приложение результирующий набор(result set)

Блоки запроса:
SELECT - единственный обязательный. Определяет столбцы, которые должны быть включены в результирующий набор
FROM - указывает таблицы, из которых должны быть получены данные и то, как это таблицы должны быть соединены
WHERE - Ограничивает число строк в окончательном результирующем наборе
GROUP BY - Используется для группировки строк по одинаковым значениям столбцов
HAVING - Ограничивает число срток, в окончательном результирующем наборе с помощью группировки строк
ORDER BY - сортирует строки окончательного результирующего набора по одному или более столбцам

Эти блоки включены в спецификацию ANSI. Кроме них в MySQL есть еще некоторые собственные блоки:
LIMIT - отбрасывается все строки в результирующем наборе, кроме первых X строк
into outfile

Блок SELECT обрабатывается одним из последних, т.к. прежде чем

формировать результирующий набор столбцов, нам надо знать, какие столбцы могут принимать участие в запросе.

В селекте можно делать много разных интересных штук, например, выполнять вычисления с результатом, использовать литералы и встроенный функции, например: SELECT emp_id, 'ACTIVE', emp_id * 3.14159, UPPER(lname) FROM employee;

Если требуется просто выполнить встроенную функцию или просто вычислить какое-то выражение, то можно обойтись без блока FROM, например, SELECT VERSION(), USER(), DATABASE() вернет вашу версию mysql, пользователя, под которым вы сейчас и используемую БД.

Столбцам в результирующем наборе можно указывать свои имена: SELECT emp_id, 'ACTIVE' status, emp_id * 3.14159 empid_x_pi, UPPER(lname) last_name_upper FROM employee;

Что бы убрать дублирующиеся данные в запросе, используется DISTINCT, например: SELECT DISTINCT cust_id FROM account;
Формирование уникальных строк требует сортировки данных, что в больших рехультирующих наборах может оказаться достаточно ресурсоемким. Использовать только тогда, когда это действительно необходимо.

Существует три типа таблиц:
- постоянные таблицы (созданные выражением create table)
- временные таблицы (строки, возвращенные подзапросом)
- виртуальные (вьюшки, созданные выражением create view)

Каждый из этих трех типов может быть включен в блок FROM запроса.

Подзапросы могут располагаться в различных частях выражения SELECT. В рамках блока FROM подзапрос выполняет функцию временной таблицы. Вот пример: SELECT e.emp_id, e.fname, e.lname FROM (SELECT emp_id, fname, lname, start_date, title FROM employee) e;

Пример создания вьюшки:
```
CREATE VIEW employee_vw AS
SELECT emp_id, fname, lname, YEAR(start_date) start_year
FROM employee;
```

Если используется несколько таблиц, то ANSI требует, что бы указывался способ из соиединения для переносимости между разными серверами БД. Пример способа соединения:
```
SELECT employee.emp_id, employee.fname, employee.lname,
department.name dept_name
FROM employee INNER JOIN department
    ON employee.dept_id = department.dept_id;
```

Вот тот же самый пример, но с использованием псевдонимов таблиц:

```
SELECT e.emp_id, e.fname, e.lname, d.name dept_name
FROM employee e INNER JOIN department d
    ON e.dept_id = d.dept_id;
```

Блок WHERE - это механизм отсеивания нежелательных строк из результирующего набора. Пример использования:

```
SELECT emp_id, fname, lname, start_date, title
FROM employee
WHERE title = 'Head Teller';
```

Условия в блоке WHERE разделяются тремя операторами: AND, OR и NOT. Условия можно группировать с помощью круглых скобок:

```
SELECT emp_id, fname, lname, start_date, title
FROM employee
WHERE (title = 'Head Teller' AND start_date > '2002 01 01')
    OR (title = 'Teller' AND start_date > '2003 01 01');
```

HAVING позволяет фильтровать данные, сгруппированные при помощи GROUP BY аналогично блоку WHERE.

Сортировка по возрастанию - ascending(ASC), по убыванию - descending(DESC). По умолчанию выполняется сортировка по возрастанию. Пример с сотрировкой:

```
SELECT account_id, product_cd, open_date, avail_balance
FROM account
ORDER BY avail_balance DESC;
```

Сортировать можно с помощью выражений. Например, если требуется отсортировать по последним трем разрядам:

```
SELECT cust_id, cust_type_cd, city, state, fed_id
FROM customer
ORDER BY RIGHT(fed_id, 3);
```

Можно сортировать, ссылаясь не на имя колонки, а на её номер:

```
SELECT emp_id, title, start_date, fname, lname
FROM employee
ORDER BY 2, 5;
```

Отсортирует по title и lname

HAVING фильтрует по группам данных.

Пример использования оператора NOT:

```
WHERE end_date IS NULL
AND NOT (title = 'Teller' OR start_date < '2003 01 01')
```

Эквивалент этого выражения:

```
WHERE end_date IS NULL
```

```
AND title != 'Teller' AND start_date >= '2003 01 01'
```

Помимо всего прочего в блоке WHERE можно использовать встроенные функции и подзапросы

Операторы в блоке WHERE: =, !=, <, >, <>, LIKE, IN и BETWEEN
Арифметические операторы, такие как +, -, * и /

Пример подзапроса: dept_id = (SELECT dept_id FROM department WHERE name = 'Loans')

!= и <> эквивалентны

Если имеется верхняя и нижняя границы диапозона, то можно использовать BETWEEN, вместо двух условий. BETWEEN эквивалентен операторам >= и <=, т.е. обе границы включаются в диапозон.
Пример BEETWEEN:

```
SELECT emp_id, fname, lname, start_date
FROM employee
WHERE start_date BETWEEN '2003 01 01' AND '2001 01 01';
```

Пример использования оператора IN:

```
SELECT account_id, product_cd, cust_id, avail_balance
FROM account
WHERE product_cd IN ('CHK','SAV','CD','MM');
```

Подзапрос:

```
SELECT account_id, product_cd, cust_id, avail_balance
FROM account
WHERE product_cd IN (
    SELECT product_cd FROM product
    WHERE product_type_cd = 'ACCOUNT'
);
```

Пример NOT IN:

```
SELECT account_id, product_cd, cust_id, avail_balance
FROM account
WHERE product_cd NOT IN ('CHK','SAV','CD','MM');
```

Для поиска слов, насинающихся с определенной последовательностью можно использовать встроенную функцию LEFT(), но она не обладает достаточной гибкостью:

```
SELECT emp_id, fname, lname
FROM employee
WHERE LEFT(lname, 1) = 'T';
```

Для поиска по маске используются два символа:
- один любой символ

% любое количество любых символов, в.т.ч. и ни одного
Для поиска по маске используется оператор LIKE:

```
SELECT lname
FROM employee
WHERE lname LIKE '_a%e%';
```

При построении SQL-запросов можно использовать регулярные выражения:

```
SELECT emp_id, fname, lname
FROM employee
WHERE lname REGEXP '^[FG]';
```

В Оракле используется оператор REGEXP_LIKE, вместо REGEXP.

NULL не равен нулю, NULL не равен NULL. Выражение на значение NULL проверяется оператором IS NULL.
Пример:

```
ELECT emp_id, fname, lname, superior_emp_id
FROM employee
WHERE superior_emp_id IS NULL;
```

Так же есть оператор IS NOT NULL

Если в колонке может присутствовать NULL, надо всегда это учитывать.
Следующий запрос вернет неправельныйй результат:

```
SELECT emp_id, fname, lname, superior_emp_id
FROM employee
WHERE superior_emp_id != 6;
```

а следующий правильный:

```
SELECT emp_id, fname, lname, superior_emp_id
FROM employee
WHERE superior_emp_id != 6 OR superior_emp_id IS NULL;
```

При работе с таблицами надо проверять, могут ли её значения содержать null

Описамие таблицы достается командой DESC

JOIN, без указания столбцов, поторым соединять, делает декартово произведение всех строк их таблиц.
Связи между таблицами указываются в подблоке ON:

```
SELECT e.fname, e.lname, d.name
FROM employee e JOIN department d
ON e.dept_id = d.dept_id;
```

Внутреннее соединение(inner join) - если определенный идентификатор есть в одной таблице, но его нет в другой, то соединения не

происходит и они не включаются в результирующий набор

Для включения строк, значений которых нет в одной из таблиц, используется OUTER JOIN (внешнее соединение)

Надо всегда указывать тип соединения и вместо JOIN писать INNER JOIN.

Если имена столбцов совпадают, то вместо ON можно использовать подблок USING:

```
SELECT e.fname, e.lname, d.name
FROM employee e INNER JOIN department d
USING (dept_id);
```

Лучше всегда пользоваться только ON

Пример запроса с соединением 3-х таблиц:

```
SELECT a.account_id, c.fed_id, e.fname, e.lname
FROM account a INNER JOIN customer c
ON a.cust_id = c.cust_id
INNER JOIN employee e
ON a.open_emp_id = e.emp_id
WHERE c.cust_type_cd = 'B';
```

Применение подзапроса в качестве таблиц:

```
SELECT a.account_id, a.cust_id, a.open_date, a.product_cd
FROM account a INNER JOIN
(SELECT emp_id, assigned_branch_id
FROM employee
WHERE start_date <= '2003 01 01'
AND (title = 'Teller' OR title = 'Head Teller')) e
ON a.open_emp_id = e.emp_id
INNER JOIN
(SELECT branch_id
FROM branch
WHERE name = 'Woburn Branch') b
ON e.assigned_branch_id = b.branch_id;
```

Рекурсивным внечним ключем называется называется ключ, ссылающийся на эту же таблицу.
Пример рекурсивного соединения:

```
SELECT e.fname, e.lname,
e_mgr.fname mgr_fname, e_mgr.lname mgr_lname
FROM employee e INNER JOIN employee e_mgr
ON e.superior_emp_id = e_mgr.emp_id;
```

При применении операции UNION и ей подобных в обеих таблицах должно быть одинаковое количество столбцов и типы данных должны быть одинаковыми.

Операторы для работы с множествами:
UNION - объединяет две таблицы, исключая дубляж
UNION ALL - объединяет таблицы, оставляя повторяющиеся данные
INTERSECT - возвращает только те данные, которые имеются в обеих таблицах
EXCEPT - возвращает все значения первой таблицы, за вычетом значений второй

Пример работы с множествами:
```
SELECT emp_id, fname, lname
FROM employee
INTERSECT
SELECT cust_id, fname, lname
FROM individual;
```

В составных запросах рекомендуется указывать одинаковые псевдонимы для всех таблиц. Оператор сортировки для всех данных указвается в конце.

Последоватльность операций с множествами имеет значение

Ф-ция char() выводит всякие разные символы.
concat() для конкатинации.
length() - возвращает длину строки. удаляет пробелы в конце строки.
position() - возвращает номер символа, с которого начинается первое вхождение

LIKE можно использовать не только в блоке where:
```
SELECT name, name LIKE '%ns' ends_in_ns
FROM department;
```

Пишет, оканчивается ли строка на "ns"

Пример concat():
```
SELECT CONCAT(fname, ' ', lname, ' has been a ',
title, ' since ', start_date) emp_narrative
FROM employee
WHERE title = 'Teller' OR title = 'Head Teller';
```

SUBSTRING() - получает подстроку.
SELECT SUBSTRING('goodbye cruel world', 9, 5); возвращает "cruel"

MOD(10,4) - возвращает остаток от деления одного числа на другое
POW(2,8) - два в восьмой степени
ceil(), floor(), round(), truncate(). В большуй сторону, в меньшую сторону, в зависимости от значения и можно указывать число разрядов, можно указывать число разрядов и остальный отбрасываются без

округления.

Устанавливаем часовой пояс на время сеанса:
```
SET time_zone = 'Europe/Zurich';
```

Функция приводит значение к определенному типу данных
```
CAST('2005 03 27 15:30:00' AS DATETIME)
```

Получение текущей даты в разных вариациях:
```
SELECT CURRENT_DATE( ), CURRENT_TIME( ), CURRENT_TIMESTAMP( );
```

Добавляет к дате определенный интервал
```
SELECT DATE_ADD(CURRENT_DATE( ), INTERVAL 5 DAY)
```

Вернет последний день марта:
```
SELECT LAST_DAY('2005 03 25');
```

Возвращает дату в нужном часовом поясе

```
CONVERT_TZ(CURRENT_TIMESTAMP( ), 'US/Eastern', 'UTC')
```

Извлекает часть даты:
```
EXTRACT(YEAR FROM '2005 03 22 22:19:05')
```

Возвращает разницу в полных днях между двумя датами
```
DATEDIFF('2005 09 05', '2005 06 22');
```

Если в запросе присутствует GROUP BY, то агрегатные функции применятются к каждой группе, а не ко всем элементам подряд

Блок GROUP BY выполняется после вычисления блока WHERE. Поэтому, что бы фильтровать по группам применяется блок HAVING

```
SELECT open_emp_id, COUNT(*) how_many
FROM account
GROUP BY open_emp_id
HAVING COUNT(*) > 4;
```

Агрегатные функции: min(), max(), avg(), count(), sum()

Неявной группой называется группа, сформированная запросом без агрегатных функций.

Считает количество только уникальных значений
```
SELECT COUNT(DISTINCT open_emp_id)
FROM account;
```

В качестве аргументов агрегатных функций могут выступать выражения:

```
SELECT MAX(pending_balance - avail_balance) max_uncleared
FROM account;
```

В агрегатных функциях, так же и в любых выражениях стоит учитывать значение NULL. Как правило, оно игнорируется.

В GROUP BY можно группировать по нескольким столбцам:

```
SELECT product_cd, open_branch_id,
SUM(avail_balance) tot_balance
FROM account
GROUP BY product_cd, open_branch_id;
```

Группировка посредством выражений:

```
SELECT EXTRACT(YEAR FROM start_date) year,
COUNT(*) how_many
FROM employee
GROUP BY EXTRACT(YEAR FROM start_date);
```

WITH ROLLUP - выводит данные и по группам групп. Например, если нужны сумма по определенной группе, то будет выведена так же сумма у всех групп. Пример:

```
SELECT product_cd, open_branch_id, SUM(avail_balance) tot_balance
FROM account
GROUP BY product_cd, open_branch_id WITH ROLLUP;
```

Если группировка делается, например, по 3-столбцам, можно обобщать данные долько по 2-м: GROUP BY a, ROLLUP(b, c)

Групповая фильтрация:

```
SELECT product_cd, SUM(avail_balance) prod_balance
FROM account
WHERE status = 'ACTIVE'
GROUP BY product_cd
HAVING SUM(avail_balance) >= 10000;
```

В блок HAVING можно включать агрегатные ф-ции, которых нет в блоке select:

```
SELECT product_cd, SUM(avail_balance) prod_balance
FROM account
WHERE status = 'ACTIVE'
GROUP BY product_cd
HAVING MIN(avail_balance) >= 1000
AND MAX(avail_balance) <= 10000;
```

Подзапросы выполняются раньше, чем выражения, котори их содержит.

Пример подзапроса:

```
SELECT account_id, product_cd, cust_id, avail_balance
```

```
FROM account
WHERE account_id = (SELECT MAX(account_id) FROM account);
```

Типы подзапросов: несвязанные - полностью независимые от внешнего запроса, связанные - ссылаются на столбцы внешнего выражения.

Подрапрос, возвращающий только одну строку с одним значением называется скалярным подзапросом. Результат такого подзапроса может использоваться только с опрераторами сравнения

Подзапрос, возвращающий несколько строк с одним значением может участвовать в опрераторах IN, NOT IN, ALL, ANY

```
SELECT emp_id, fname, lname, title
FROM employee
WHERE emp_id IN (SELECT superior_emp_id
FROM employee);
```

Оператор ALL применяет операцию сравнения ко всем элементам из списка:

```
SELECT emp_id, fname, lname, title
FROM employee
WHERE emp_id <> ALL (SELECT superior_emp_id
FROM employee
WHERE superior_emp_id IS NOT NULL);
```

Надо учитывать, что в результирующем наборе может быть null. Сравнение с null возвратит пустой набор:

```
SELECT emp_id, fname, lname, title
FROM employee
WHERE emp_id NOT IN (1, 2, NULL);
```

Пример с ALL:

```
SELECT account_id, cust_id, product_cd, avail_balance
FROM account
WHERE avail_balance < ALL (SELECT a.avail_balance
FROM account a INNER JOIN individual i
ON a.cust_id = i.cust_id
WHERE i.fname = 'Frank' AND i.lname = 'Tucker');
```

С ALL находит все счата, где доступный баланс меньшье, чем самый маленький.
ANY находит все счета, где доступный баланс меньше, чем самый большой

Подзапрос с двумя столбцами:

```
SELECT account_id, product_cd, cust_id
FROM account
WHERE (open_branch_id, open_emp_id) IN
(SELECT b.branch_id, e.emp_id
```

```
FROM branch b INNER JOIN employee e
ON b.branch_id = e.assigned_branch_id
WHERE b.name = 'Woburn Branch'
AND (e.title = 'Teller' OR e.title = 'Head Teller'));
```

Связанный подзапрос выполняется для каждой строки-кандидата:

```
SELECT c.cust_id, c.cust_type_cd, c.city
FROM customer c
WHERE 2 = (SELECT COUNT(*)
FROM account a
WHERE a.cust_id = c.cust_id);
```

Оператор EXISTS проверяет, существует ли связь вообще, несчитая количество найденных элементов:

```
SELECT a.account_id, a.product_cd, a.cust_id, a.avail_balance
FROM account a
WHERE EXISTS (SELECT 1
FROM transaction t
WHERE t.account_id = a.account_id
AND t.txn_date = '2005 01 22');
```

При использовании EXISTS принято задовать SELECT 1 или SELECT *

Для поиска подзапросов, не возвращающих строки можно использовать NOT EXISTS

Подзапросы можно использовать и с операторами INSERT, UPDATE, DELETE:

```
UPDATE account a
SET a.last_activity_date =
(SELECT MAX(t.txn_date)
FROM transaction t
WHERE t.account_id = a.account_id);
```

В MySQL при использовании связанных подзапросов не допускаются псевдонимы таблиц.

Использование подзапроса в блоке FROM:

```
SELECT d.dept_id, d.name, e_cnt.how_many num_employees
FROM department d INNER JOIN
(SELECT dept_id, COUNT(*) how_many
FROM employee
GROUP BY dept_id) e_cnt
ON d.dept_id = e_cnt.dept_id;
```

Подзапрос в блоке HAVING:

```
SELECT open_emp_id, COUNT(*) how_many
FROM account
GROUP BY open_emp_id
```

```
HAVING COUNT(*) = (SELECT MAX(emp_cnt.how_many)
FROM (SELECT COUNT(*) how_many
FROM account
GROUP BY open_emp_id) emp_cnt);
```

Можно использовать подзапросы в ORBER BY для целей сортировки.

Пример внешнего рекурсивного запроса:

```
SELECT e.fname, e.lname,
e_mgr.fname mgr_fname, e_mgr.lname mgr_lname
FROM employee e LEFT OUTER JOIN employee e_mgr
ON e.superior_emp_id = e_mgr.emp_id;
```

Если необходимо получить декартово произведение, надо выполнить перекрестное соединение:

```
SELECT pt.name, p.product_cd, p.name
FROM product p CROSS JOIN product_type pt;
```

Для определения условной логики служит оператор CASE:

```
SELECT c.cust_id, c.fed_id,
CASE
WHEN c.cust_type_cd = 'I'
THEN CONCAT(i.fname, ' ', i.lname)
WHEN c.cust_type_cd = 'B'
THEN b.name
ELSE 'Unknown'
END name
FROM customer c LEFT OUTER JOIN individual i
ON c.cust_id = i.cust_id
LEFT OUTER JOIN business b
ON c.cust_id = b.cust_id;
```

Кроме выражения CASE в субд есть функции имитирующие условныйе операторы. В MySQL - if().
CASE поддерживают все СУБД.

Это было выражение case с переборором вариантов:

```
CASE
WHEN C1 THEN E1
WHEN C2 THEN E2
...
WHEN CN THEN EN
[ELSE ED]
END
```

Все выражения, возвращаемые блоками WHEN должны возвращать результаты одного типа.

Простое выражение CASE:

```
CASE V0
WHEN V1 THEN E1
WHEN V2 THEN E2
...
WHEN VN THEN EN
[ELSE ED]
END
```

Типа как case в php:

```
CASE customer.cust_type_cd
WHEN 'I' THEN
(SELECT CONCAT(i.fname, ' ', i.lname)
FROM individual I
WHERE i.cust_id = customer.cust_id)
WHEN 'B' THEN
(SELECT b.name
FROM business b
WHERE b.cust_id = customer.cust_id)
ELSE 'Unknown Customer Type'
END
```

Проверка существования:

```
SELECT c.cust_id, c.fed_id, c.cust_type_cd,
CASE
WHEN EXISTS (SELECT 1 FROM account a
WHERE a.cust_id = c.cust_id
AND a.product_cd = 'CHK') THEN 'Y'
ELSE 'N'
END has_checking,
CASE
WHEN EXISTS (SELECT 1 FROM account a
WHERE a.cust_id = c.cust_id
AND a.product_cd = 'SAV') THEN 'Y'
ELSE 'N'
END has_savings
FROM customer c;
```

```
SELECT 100 / 0;
```

MySQL вернет null, Oracle выдаст ошибку

Что бы уберечься от деления на ноль, можно использовать условный оператор.

Вычисления с NULL равны NULL

Запросы на чтение блокируются до тех пор, пока не будет снята

блокировка записи, либо, при контроле версий, чтение и запись не зависят друг от друга.

Блокировка может выполняться на одном из трех уровней детализации (granularities):
- блокировка таблицы.
- блокировка страницы. предотвращает одновременную запись в одну страницу таблицы (страница - сенмент памяти, обычно от 2 до 16 КБайт)
- блокирование строки

При совершении операции, пользователь получит данные, которые были на начало содания отчета, если сервер использует транзакции и на момент созданиея блокировки для блокировок.

В Oracle даже одиночные запросы запускаются с транзакцией и если результать неудовлетворительный, то операцию можно откатить.

MySQL позволяет отключить режим автоматической фиксации так:
SET AUTOCOMMIT=0
Если режим автоматической фиксации выключен, то все операции по фиксации и откату надо выполнять явно.
Автор книги рекомендует отключать автоматическую фискацию перед началом работы.

Запросы управления схемой (alter table) приводят к автоматической фиксации транзакции.

Если запустить еще одну команду start transaction, то предыдущая автоматом фиксируется.

В транзакциях можно использовать точки созранения, что откатываться к определенному месту, а не в самое начало транзакции:
SAVEPOINT my_savepoint;
ROLLBACK TO SAVEPOINT my_savepoint;

MyISAM - нетранзакционный, использует блокировку таблицы
InnoDb - транзакционный, использует блокировку строки

Покажет информацию о таблице:
SHOW TABLE STATUS LIKE 'transaction'

```
START TRANSACTION;
UPDATE product
SET date_retired = CURRENT_TIMESTAMP( )
WHERE product_cd = 'XYZ';
SAVEPOINT before_close_accounts;
UPDATE account
```

```
SET status = 'CLOSED', close_date = CURRENT_TIMESTAMP( ),
last_activity_date = CURRENT_TIMESTAMP( )
WHERE product_cd = 'XYZ';
ROLLBACK TO SAVEPOINT before_close_accounts;
COMMIT;
```

Добавление индекса:

```
ALTER TABLE department
ADD INDEX dept_name_idx (name);
```

Просмотр индексов:

```
SHOW INDEX FROM department
```

b-tree - сбалансированное дерево. Индексы на основе b-tree - по умолчанию.
b-tree подходят для таблиц с множеством различных значений.
Оракл предоставляет возможность создания битового индекса, которые хорош для большого количества часто повторяющихся данных.
Существуют так же текстовые индексы

Оператор explain не выполняет запрос, а только показывает план его выполнения:

```
EXPLAIN SELECT cust_id, SUM(avail_balance) tot_bal
FROM account
WHERE cust_id IN (1, 5, 9, 11)
GROUP BY cust_id
```

Ограничения:
- Ограничения первичного ключа (Primary-key constraints): идентифицирует столбец или столбцы, гарантируя уникальность
- Ограничения внешнего ключа (Foreign-key constraints): значения могут содержать только значения первичного ключа другой таблицы
- Ограничения уникальности (Unique constraints)
- Проверочные ограничения целостности: ограничивают допустимые значения

В MySQL для ограничений внешнего ключа используется InnoDB

Ограничения:

```
CREATE TABLE product
(product_cd VARCHAR(10) NOT NULL,
name VARCHAR(50) NOT NULL,
product_type_cd VARCHAR (10) NOT NULL,
date_offered DATE,
date_retired DATE,
CONSTRAINT fk_product_type_cd FOREIGN KEY (product_type_cd)
REFERENCES product_type (product_type_cd),
```

```
CONSTRAINT pk_product PRIMARY KEY (product_cd)
);
```

Добавление ограничения к уже существующие таблице:

```
ALTER TABLE product
ADD CONSTRAINT pk_product PRIMARY KEY (product_cd);
ALTER TABLE product
ADD CONSTRAINT fk_product_type_cd FOREIGN KEY (product_type_cd)
REFERENCES product_type (product_type_cd);
```

Для удаления ограничений используется ключение слово drop, а не add

При ограничениях внешнего ключа огранизуется двусторонняя целостность, т.е. нельзя переименова или удалить ключ родительской таблицы без соответствующих изменения в дочерних и нельзя добавлять в дочернюю внешнего ключа, которого нет в родительской.

MySQL позволяет записывать результаты запроса в файл:

```
SELECT emp_id, fname, lname, start_date
INTO OUTFILE 'C:\\TEMP\\emp_list.txt'
FROM employee;
```

Любопытный инсерт:

```
INSERT INTO login_history (cust_id, login_date)
SELECT c.cust_id,
ADDDATE(a.open_date, INTERVAL a.account_id * c.cust_id HOUR)
FROM customer c CROSS JOIN account a;
Query OK, 312 rows affected (0.03 sec)
```

Создание клона таблицы:

```
CREATE TABLE individual2 AS
SELECT * FROM individual;
```

Удаление данных из нескольких таблиц:

```
DELETE account2, customer2, individual2
FROM account2 INNER JOIN customer2
ON account2.cust_id = customer2.cust_id
INNER JOIN individual2
ON customer2.cust_id = individual2.cust_id
WHERE individual2.cust_id = 1;
```