Основы SQL

Введение в язык SQL и выполнение запросов для извлечения данных

Основные СУБД

Введение в SQL

SQL (англ. Structured Query Language – «язык структурированных запросов») – универсальный компьютерный язык, применяемый для создания, модификации и управления данными в реляционных базах данных.

- **База данных** – это организованное хранилище данных. БД состоит из одной или нескольких таблиц.
- **Система управления базами данных (СУБД)** – специальный тип программного обеспечения, управляющий хранилищами данных.

Преимущества SQL

- **Непроцедурный язык**
  - обрабатывает одновременно множество записей
  - обеспечивает автоматическое управление данными
- **Язык для всех пользователей**
  - используется во всех типах действий с базой данных всеми категориями пользователей
- **Унифицированный язык**
  - обеспечивает операции для самых различных задач, включая: выборку данных, добавление, изменение и удаление строк из таблицы, создание, модификацию и удаление объектов базы данных, управление доступом к базе данных и объектам базы, гарантирование согласованности и целостности данных
- **Общий язык для всех реляционных БД**

Таблица

Таблица – это главный объект БД.

- В таблицах хранится информация.
- Таблица представляет из себя совокупность столбцов.

Столбец(column) – содержит информацию одного типа.

Строка(row) – горизонтальный ряд ячеек, отведенный для каждого объекта таблицы.

Запись (record) – данные в строке.

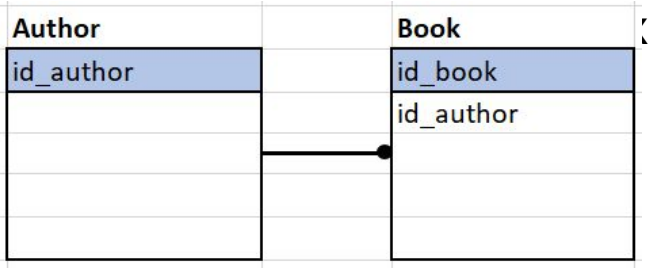
Поле(field) – пересечение столбца и строки.

	AIRCRAFT_CODE	MODEL	FARE_CONDITIONS	COUNT(*)
1	CN1	Cessna 208 Caravan	Economy	12
2	CR2	Bombardier CRJ-200	Economy	50
3	SU9	Sukhoi Superjet-100	Business	12
4	SU9	Sukhoi Superjet-100	Economy	85
5	319	Airbus A319-100	Business	20
6	319	Airbus A319-100	Economy	96

Целостность данных

Базы данных содержат таблицы, которые связаны между собой различными связями. Связь (relationship) представляет ассоциацию между сущностями разных типов.
Выделяют главную или родительскую таблицу (primary key table) и зависимую, дочернюю таблицу (foreign key table). Дочерняя таблица зависит от родительской.
Для организации связи используются внешние ключи. Внешний ключ представляет один или несколько столбцов из одной таблицы, который одновременно является потенциальным ключом из другой таблицы. Как правило, внешний ключ из зависимой таблицы указывает на первичный ключ из главной таблицы.

Первичный ключ (primary key) - столбец или комбинация столбцов уникально идентифицирующий каждую запись в таблице. Первичный ключ должен всегда принимать уникальное значение и не может быть NULL
Внешний ключ (foreign key) – используется для отражения связи между таблицами. Это столбец или комбинация столбцов, значения которых соответствуют Первичному ключу в родительской таблице.
Ссылочная целостность – в подчиненных таблицах не должно быть записей, ссылающихся на несу таблиц.



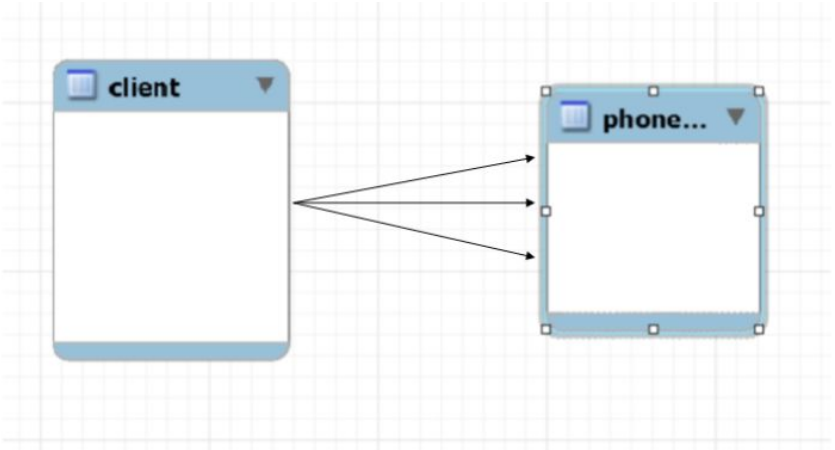
Author

id_author	name
1	Пушкин
2	Толстой
3	Гоголь

Book

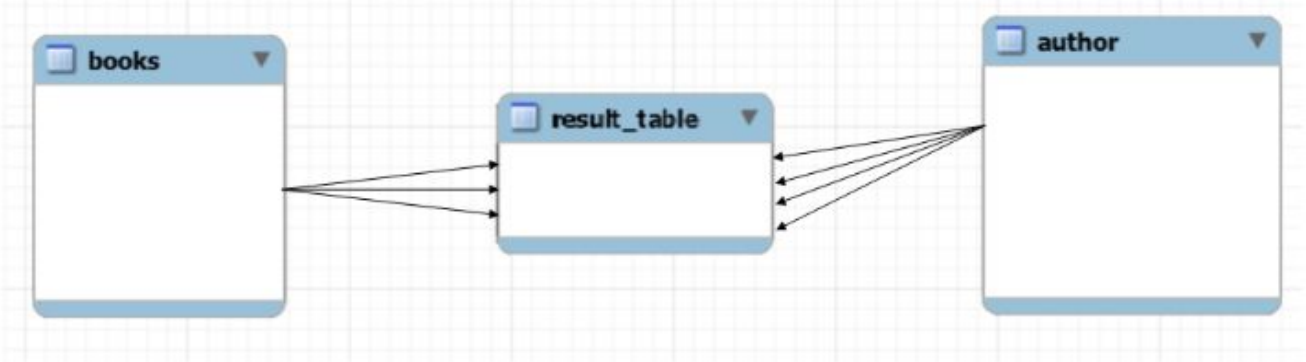
id_book	id_author	name
11	1	Евгений Онегин
15	1	Руслан и Людмила
7	2	Война и мир
5	4	Преступление и наказание

Связь один ко многим в базах данных реализуется тогда, когда объекту А может принадлежать или же соответствовать несколько объектов Б, но объекту Б может соответствовать только один объект А



У одного клиента может быть несколько номеров

Связь многие ко многим реализуется в том случае, когда нескольким объектам из таблицы А может соответствовать несколько объектов из таблицы Б, и в тоже время нескольким объектам из таблицы Б соответствует несколько объектов из таблицы А. Рассмотрим простой пример.



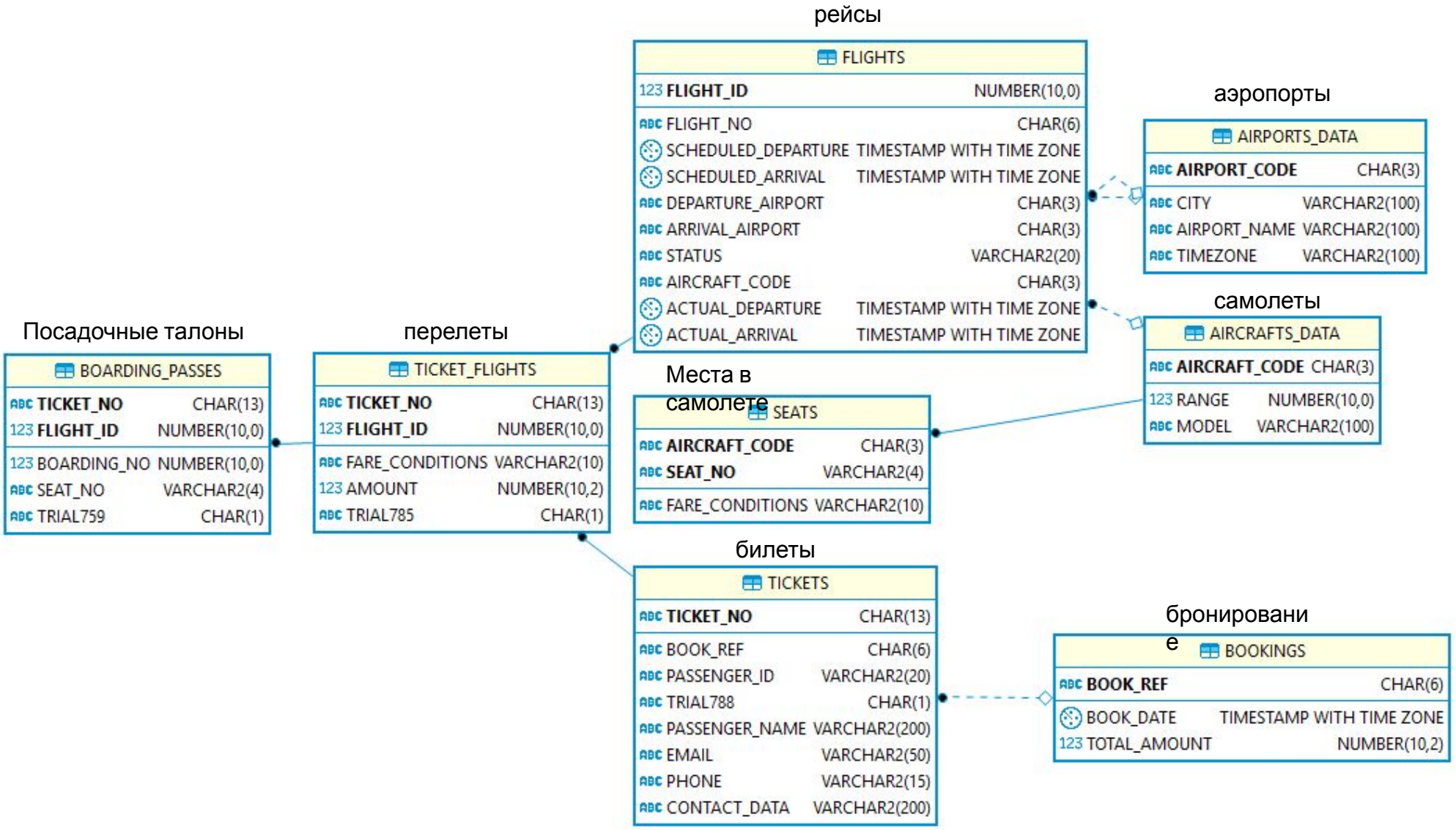
Одна книга могла быть написана несколькими авторами.
Автор мог написать несколько книг.

Связь один к одному

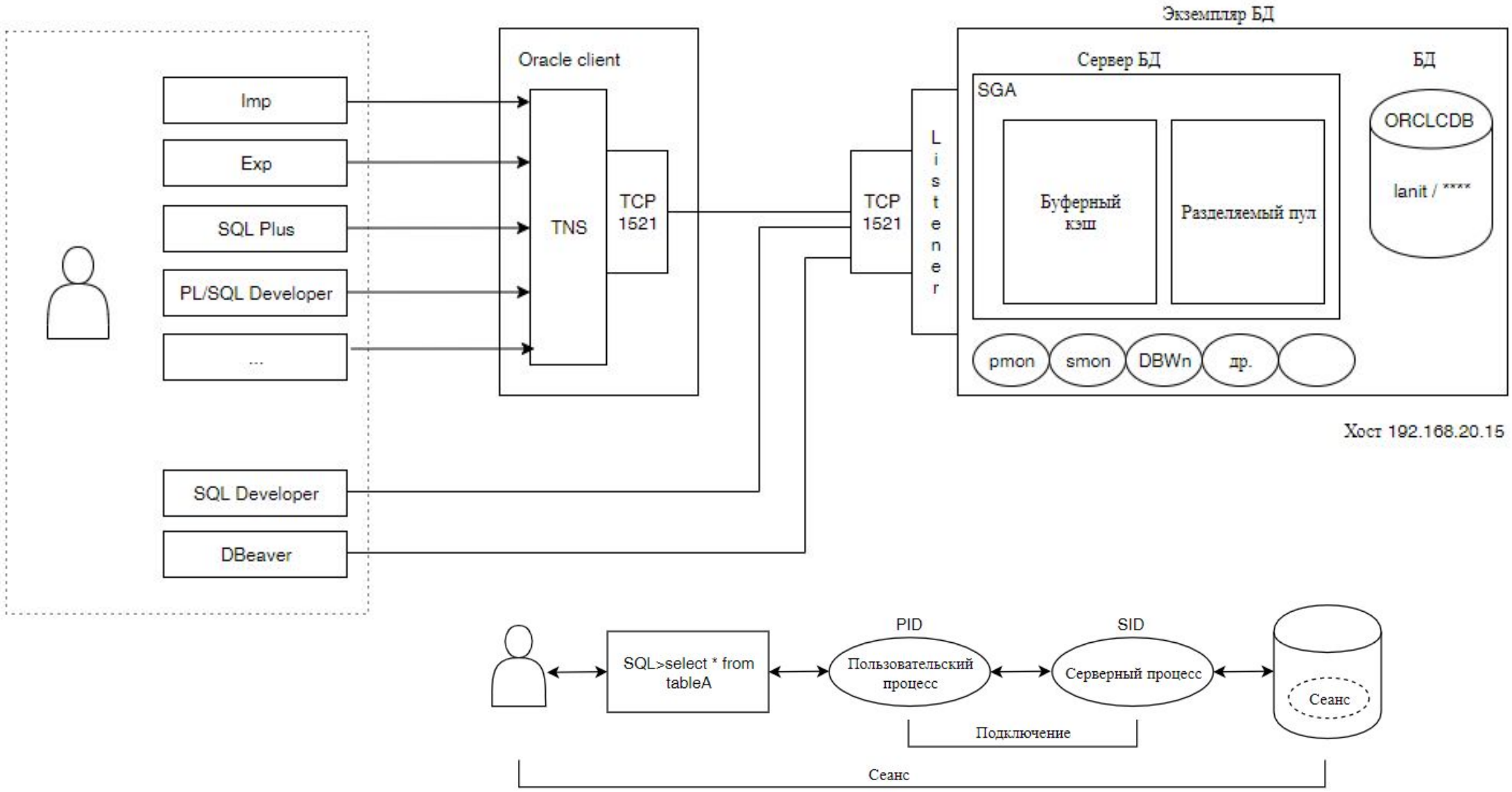
В редких случаях связь один-к-одному моделируется используя две таблицы.
Такой вариант иногда необходим для увеличения производительности (например, иногда — это вынесение поля с типом данных blob в отдельную таблицу для ускорения поиска по родительской таблице). Или порой вы можете решить, что вы хотите разделить две сущности в разные таблицы в то время, как они все еще имеют связь один-к-одному.
Но обычно наличие двух таблиц в связи один-к-одному считается дурной практикой!

Пример:
Сотрудники – Паспортные данные
Документы(основные данные) – Документы (дополнительные данные)

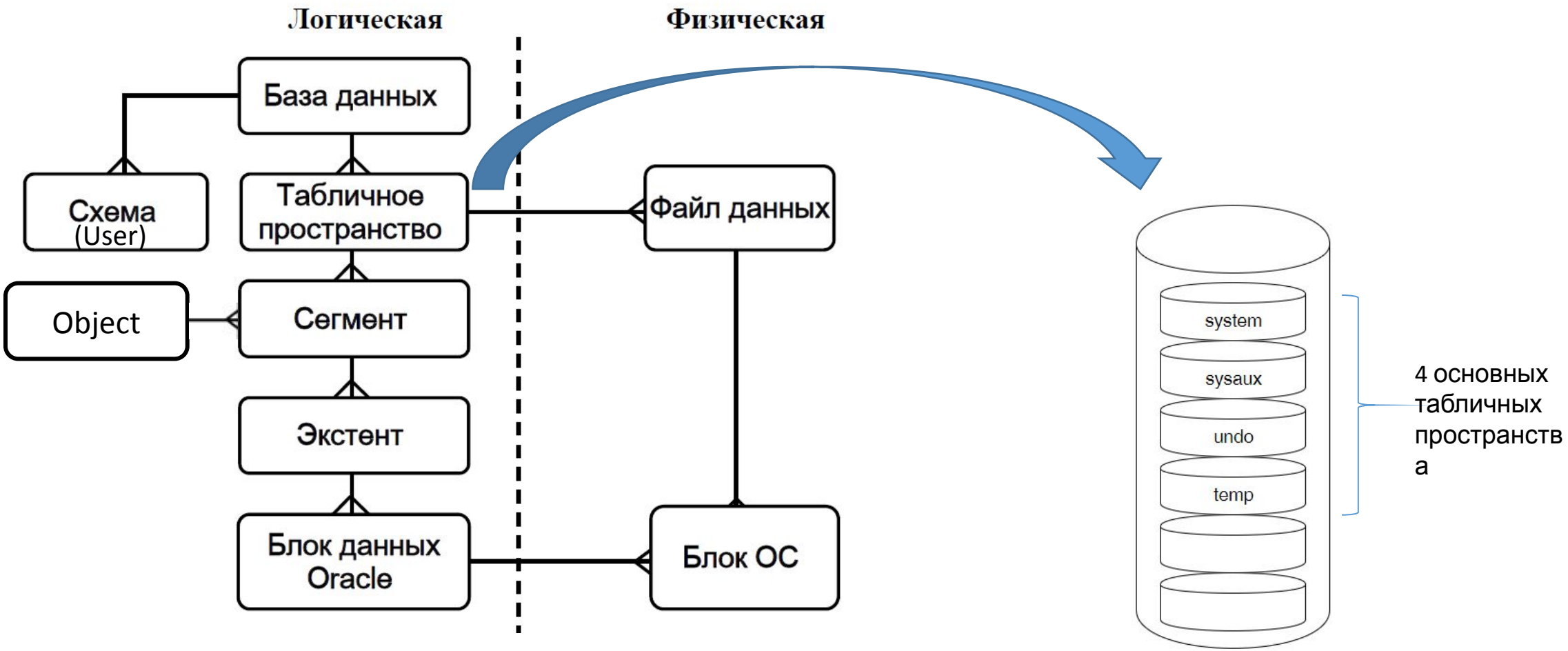
База данных «Авиаперевозки»



Подключение к БД



Логическая и физическая структура БД



- База данных
  - Пользователь (схема)
  - anna
    - Объект – таблица A
    - Объект – таблица B
    - Объект – процедура P1
  - Пользователь (схема)
  - salary
    - Объект – таблица C
    - Объект – триггер T1
  - Пользователь (Схема)
  - pir
    - Объект – таблица A
    - Объект- таблица F

Объекты БД

Таблицы представляют собой сегменты базы данных, в которых хранятся собственно данные. Каждая таблица состоит из строк (записей). Каждый столбец таблицы имеет имя и содержит данные одного типа.

	AIRPORT_CODE	CITY	AIRPORT_NAME	TIMEZONE
1	KHV	Khabarovsk	Khabarovsk-Novy Airport	Asia/Vladivostok
2	YKS	Yakutsk	Yakutsk Airport	Asia/Yakutsk
3	MJZ	Mirnyj	Mirny Airport	Asia/Yakutsk
4	PKC	Petropavlovsk	Yelizovo Airport	Asia/Kamchatka
5	UUS	Yuzhno-Sakhalinsk	Yuzhno-Sakhalinsk Airport	Asia/Sakhalin
6	VVO	Vladivostok	Vladivostok International Airport	Asia/Vladivostok
7	LED	St. Petersburg	Pulkovo Airport	Europe/Moscow

Представления - Виртуальная (логическая) таблица, представляющая собой поименованный запрос (синоним к запросу), который будет подставлен как подзапрос при использовании представления.)

```
CREATE OR REPLACE VIEW V_AIRPORTS AS
SELECT airport_code, airport_name, city, ml.timezone
FROM airports_data t
WHERE t.airport_code in ('320','321','733', 'CR2');
```

select * from V_AIRPORTS

Синонимы - псевдонимы объектов базы данных, которые служат в основном для облегчения пользователям доступа к объектам, принадлежащим другим пользователям, а также в целях безопасности.

Последовательности - объект базы данных, который генерирует целые числа в соответствии с правилами, установленными во время его создания. Для последовательности можно указывать как положительные, так и отрицательные целые числа

Хранимая процедура – программа, написанная на языке PL/SQL, которая выполняет некоторые действия с информацией в базе данных и при этом сама хранится в базе данных.

Функции PL/SQL - Похожа на процедуру PL/SQL: она также имеет спецификацию и тело. Главное различие между процедурой и функцией в том, что функция предназначена для возврата значения, которое может использоваться в более крупном SQL-Операторе

Пакет Oracle PL/SQL - Объект схемы, который группирует логически связанные типы, процедуры и функции. Пакеты обычно состоят из двух частей: спецификации и тела

Триггер - программа на языке PL/SQL, которая срабатывает всякий раз, когда происходит событие, которое вызывает триггер.

Индексы - обеспечивают быстрый доступ к строкам таблиц, сохраняя отсортированные значения указанных столбцов и используя эти отсортированные значения для быстрого нахождения ассоциированных строк таблицы.

Типы данных (SQL типы используются для хранения данных в БД)

Символьные	**VARCHAR2(n)** – строки переменной длины, т.е. требует памяти столько, сколько данных **CHAR(n)** – строки постоянной длины, т.е. ввели меньше данных в строку – размер не изменится (0-256). Если значение короче, оно дополняется пробелами. Лучше не использовать **CLOB** (Character Large Object) - большой символьный объект (до 4ГБайт)
Числовые	**NUMBER (m,n)** – используется для хранения чисел с фиксированной и плавающей точкой. M - ТОЧНОСТЬ (общее число цифр), n – МАСШТАБ (число цифр справа от десятичной точки). По умолчания 10 знаков. **INTEGER** – число без десятичной точки. Oracle исправит на number
Дата/время	**DATE** - дата в формате yyyy-mm-dd (ISO), dd/mm/yyyy (ANSI), dd-MON-yy. **INTERVAL** – расстояние между датами
Другие	**XMLTYPE** – хранения XML документов **BLOB** (Binary Large OBject) - Двоичный большой объект. Для хранения изображений, аудио, видео.
Булевы	нет, используется number(1)

Также значением поля может быть NULL – означает отсутствие значений – пустую ячейку.

Структура языка SQL

DML(Data Manipulation Language) операторы манипуляции данными:

- SELECT считывает данные, удовлетворяющие заданным условиям,
- INSERT добавляет новые данные,
- UPDATE изменяет существующие данные,
- DELETE удаляет данные;

DDL(Data Definition Language) операторы определения данных:

- CREATE создает объект БД (саму базу, таблицу, представление, пользователя и т. д.),
- ALTER изменяет объект,
- DROP удаляет объект;

TCL(Transaction Control Language) операторы управления транзакциями:

- COMMIT применяет транзакцию,
- ROLLBACK откатывает все изменения, сделанные в контексте текущей транзакции,
- SAVEPOINT делит транзакцию на более мелкие участки.

DCL(Data Control Language) операторы определения доступа к данным:

- GRANT предоставляет пользователю (группе) разрешения на определенные операции с объектом,
- REVOKE отзывает ранее выданные разрешения,
- DENY задает запрет, имеющий приоритет над разрешением;

Команда SELECT

SELECT (англ., означает «выбрать») - оператор SQL, возвращающий набор данных (выборку) из базы данных.

Общий синтаксис:

SELECT [{ ALL | DISTINCT }] { *список_вывода* | * }

FROM *имя_таблицы1* [*синоним1*] [, *имя_таблицы2* [*синоним2*].,..]

[**WHERE** *условие_отбора_записей*]

[**GROUP BY** { *имя_поля* | *выражение* }.,.. *группировка_записей*]

[**HAVING** *условие_отбора_групп*]

[**UNION** [**ALL**] SELECT … *оператор_объединения*]

[**ORDER BY** *имя_поля1* | *целое* [**ASC** | **DESC**]

[, *имя_поля2* | *целое* [**ASC** | **DESC**].,..] *оператор_сортировки_записей*];

Элементы команды SELECT

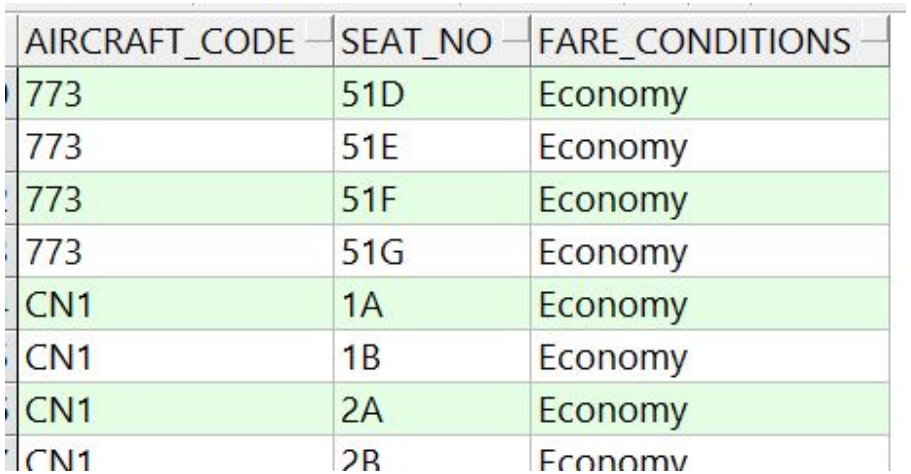
- **ALL** (действует по умолчанию) – обеспечивает включение в результаты запроса и повторяющихся значений.
- **DISTINCT** – запрещает появление строк-дублей в выходном множестве.
- **FROM** – определяются имена используемой таблицы или нескольких таблиц. Для переопределения имени результирующего столбца (создания его синонима) используется ключевое слово **AS**.
- **WHERE** – накладывается условие отбора данных.
- **GROUP BY** – образуются группы строк, имеющие одинаковые значения в указанном столбце.
- **HAVING** – накладывается условие на отбор сгруппированных строк. Группы, не удовлетворяющие условному выражению, приведенному в разделе HAVING, исключаются.
- **SELECT** – определяются столбцы, которые нужно отобразить в результате.
- **ORDER BY** – отобранные данные сортируются по указанным столбцам.

Примеры простых запросов

```
select * from seats t


select t.* from lanit.seats t


select t.*
        , t.seat_no
        , t.fare_conditions cond
        , t.fare_conditions cond2
        , t.seat_no||' '|| t.fare_conditions as cond3
from seats t
order by 1, t.fare_conditions,  t.seat_no desc
```



AIRCRAFT_CODE	SEAT_NO	FARE_CONDITIONS
773	51D	Economy
773	51E	Economy
773	51F	Economy
773	51G	Economy
CN1	1A	Economy
CN1	1B	Economy
CN1	2A	Economy
CN1	2B	Economy

Выбор данных из таблицы

WHERE – содержит условия выбора отдельных записей. Условие является логическим выражением и может принимать одно из 3-х значений:

- TRUE – истина,
- FALSE – ложь,
- NULL – неизвестное, неопределённое значение (интерпретируется как ложь).

Только те строки, для которых условное выражение возвращает значение TRUE, включаются в результат.
Условие формируется путём применения различных операторов и предикатов.

- **Операторы сравнения:**
  = равно, <>, != не равно, > больше,
  >= больше или равно, <= меньше или равно, < меньше.
- **Булевы условия:** AND, OR, NOT
- **Диапазон значений:** BETWEEN/NOT BETWEEN
- **Принадлежность к множеству:** IN/NOT IN
- **Соответствие шаблону:** LIKE/NOT LIKE
- **Пустое значение:** IS NULL/IS NOT NULL
- **Многократное сравнение:** ANY/ALL
- **Существование:** EXISTS/NOT EXISTS

GROUP BY

GROUP BY — используется для объединения строк с общими значениями.

ORDER BY

ORDER BY - используется для сортировки записей.
DESC - по убыванию
ASC - по возрастанию (по умолчанию)

Пример. Выборка списка воздушных средств с сортировкой по коду ТС в обратном порядке:

```
select *
from lanit.aircrafts_data t
order by aircraft_code desk
```

```
select aircraft_code, range, model
from lanit.aircrafts_data t
order by 1 desc
```

Примеры

Выборка из таблицы

```
select * from seats t
select t.seat_no from seats t
```

Выборка без дубликатов

```
select distinct t.seat_no from seats t
select distinct t.fare_conditions from seats t
```

Вывести список всех мест в самолете с кодом 773

```
select t.aircraft_code
        , t.seat_no
        , t.fare_conditions
from lanit.seats t
where t.aircraft_code='773'
order by t.seat_no
```

Список билетов для бронирования с кодом 969A1D

```
select ticket_no, passenger_id, passenger_name
from tickets t
where t.book_ref='969A1D'
```

Использование IN. Вывести список всех мест бизнес класса в самолетах с кодом 773, 321

```
select aircraft_code, seat_no, fare_conditions
from lanit.seats t
where t.aircraft_code in ('773','321')
        and t.fare_conditions='Business‘
```

Использование not IN. Вывести список всех мест бизнес класса кроме самолетов с кодом 773, 321

```
select aircraft_code, seat_no, fare_conditions
from lanit.seats t
where t.aircraft_code not in ('773','321')
        and t.fare_conditions='Business‘
```

Использование BETWEEN. Вывести все данные о самолетах дальностью полета от 5000 до 7000км.

```
select aircraft_code, range, model
from lanit.aircrafts_data t
where t.range between 5000 and 7000
```

Оператор LIKE и not LIKE

Позволяет выполнять сопоставление с шаблоном.

expression LIKE pattern [ESCAPE ‘escape_character’]

Подстановочный символ «_» - Соответствует одному символу
Подстановочный символ «%» - Соответствует любой строке любой длины (в том числе нулевой длины)

Найти все самолеты начинающиеся на “BO”

```
select aircraft_code, range, model
from lanit.aircrafts_data t
where upper(t.MODEL) like 'BO%‘
```

Найти все самолеты, код которых не заканчивается на 3

```
select aircraft_code, range, model
from lanit.aircrafts_data t
where t.aircraft_code not like ‘%3’
```

BETWEEN условие

Условие **BETWEEN** (называемое, также оператор **BETWEEN**) используется для получения значений в пределах диапазона в предложениях SELECT, INSERT, UPDATE или DELETE. Возвращает записи, где expression находится в пределах диапазона от value1 до value2 (включительно).

expression {not} BETWEEN value1 AND value2

Найти все рейсы с 01.09.17 по 03.09.17 включительно

```
select * from FLIGHTS t
where trunc(t.date_arrival) between to_date('01.09.2017','dd.mm.yyyy')
        and to_date('02.03.2017','dd.mm.yyyy')
order by t.date_arrival
```

Проверить, что вернет следующий запрос

```
select * from FLIGHTS t
where t.date_arrival between to_date('01.09.2017','dd.mm.yyyy')
        and to_date('02.03.2017','dd.mm.yyyy')
order by t.date_arrival
```

Выборка вычисляемых значений

Вывести дальность полета всех самолетов в тыс.км

```
select aircraft_code
        , round(range/1000,0) range
        , model
from lanit.aircrafts_data t
```

! Вывести все коды самолетов, в которых есть места бизнес класса

! Вывести все данные о самолетах с номерами начинающимися на 7, 3

! Вывести все места эконом класса с буквами A,C или места бизнес класса в самолете 773

Системная таблица DUAL

В некоторых случаях может потребоваться вернуть при помощи запроса результат работы некоторой хранимой функции или результат вычисления. В этом случае можно использовать специальную системную таблицу DUAL, доступную всем пользователям и всегда содержащую единственный столбец с именем DUMMY и типом VARCHAR2(1) и единственную строку.

Таблица DUAL - это реальная таблица в схеме SYS, содержащая только одну запись.

Пример запроса к таблице DUAL:

- select 4 + 5*20 from dual
- select sysdate from dual
- select function1(x) from dual
- select dbms_random.value from dual -- случайное значение

Конкатенированные столбцы

Оператор конкатенации - две вертикальные черты (| |):

```
select 'Данные '||aircraft_code||' '||seat_no||' '||fare_conditions
from lanit.seats t
```

NULL - выражения и функция NVL

Вычисляемые поля

- Если в поле нет значения, оно считается пустым (NULL).
- Пустое значение - это значение, которое недоступно, не присвоено, неизвестно или не определено.
- Если хотя бы одно поле в выражении имеет пустое значение, то и результат этого выражения неопределенен.
- Чтобы получить результат, необходимо преобразовать пустые значения. Для этого используется функция NVL, преобразующая пустые значения в непустые **NVL(**строка, значение)

```
select fio_human
      , sal_osn
      , sal_add
      , sal_osn+sal_add sual_all
      , sal_osn+nvl(sal_add,0) sal_all2
from SALARY t
```

	FIO_HUMAN	SAL_OSN	SAL_ADD	SUAL_ALL	SAL_ALL2
1	Иванов И.И	50000,00			50000
2	Петров П.П.	80000,00	15000,00	95000	95000
3	Ягодов Я.Я.	75000,00			75000

Форматирование даты to_char()

Параметр для to_char()	Описание
YYYY	4-значный год
YYY, YY, Y	Последние 3,2,1 цифры года
MM	Месяц (01-12; JAN = 01)
MON	Сокращенное название месяца
MONTH	Название месяца, дополненное пробелами длиной до 9 символов
D	День недели (1-7)
DD	День месяца (1-31)
DDD	День года (1-366)
DAY	Название дня
HH или HH12	Час дня (1-12)
HH24	Час дня (0-23)
MI	Минуты (0-59)
SS	Секунды (0-59)
Q	Квартал года (1, 2, 3, 4; JAN-MAR = 1)

Функции для работы со строками

Функция	Описание
INSTR	возвращает n-е вхождение подстроки в строке SELECT INSTR('На дворе трава', 'а') FROM DUAL;
LENGTH	возвращает длину указанной строки.
LOWER	преобразует все символы в заданной строке в нижний регистр
UPPER	преобразует все символы строки в верхний регистр
TRIM	удаляет все указанные символы с начала или окончания строки
TO_CHAR	преобразует число или дату в строку
SUBSTR	позволяет извлекать подстроку из строки SUBSTR(string, start_position, [length]) SELECT SUBSTR('КакЧертИзТабакерки', 1, 3) FROM DUAL; SELECT SUBSTR('КакЧертИзТабакерки', -15, 4) FROM DUAL;
REPLACE	заменяет последовательность символов в строке другим набором символов REPLACE(string1, string_to_replace, [replacement_string]) SELECT REPLACE('222abcd', '2', '3') FROM DUAL;
LPAD (RPAD)	добавляет с левой (правой) части строки определенный набор символов LPAD(string1, padded_length, [pad_string]) SELECT LPAD('lpad', 8, '0') FROM DUAL;

TRUNC ФУНКЦИЯ (ДЛЯ ДАТ)

функция TRUNC возвращает дату, усеченную к определенной единице измерения

TRUNC (date, [format])

Unit	Действительные параметры формата
Год	YYYY, YEAR, SYEAR, YYY, YY, Y
Квартал	Q
Месяц	MONTH, MON, MM
Неделя	WW
IW	IW
W	W
День	DDD, DD
День начала недели	DAY, DY, D
Час	HH, HH12, HH24
Минута	MI

Выборка данных из нескольких таблиц

Декартово произведение - соединение без конструкции WHERE, в результате которого каждая строка одной таблицы комбинируется с каждой строкой другой таблицы.

SELECT table1.columnA, table2. columnB

FROM table1, table2;

Выборка данных из нескольких таблиц с условием:

SELECT table1.columnA, table2. columnB

FROM table1, table2

WHERE table1.id_table1 =table2.id_table1

Выборка всех данных по самолетам и мест в них

```
select a.* -- выборка все столбцов таблицы aircrafts_data
        , s.seat_no
from lanit.aircrafts_data a
     , seats s
where a.aircraft_code=s.aircraft_code
order by a.aircraft_code, s.seat_no


select a.*, s.seat_no
from lanit.aircrafts_data a
     , seats s
where a.aircraft_code=s.aircraft_code(+)
order by a.aircraft_code, s.seat_no
```

При использовании жесткого равенства, в списке не будет самолетов, у которых нет посадочных мест. Если такие данные нужны, то используем (+), т.е. данные в таблице seats, могут быть или не быть. Лучше использовать left join.

Rownum – выбор первых N записей из таблицы

Сейчас 01.09.2017 07:55:00, выберите ближайшие 5 рейсов из Санкт-Петербурга в Москву

```
select t.*
from (select f.date_departure
          , a.city
              , a_ar.city city_arrival
    from flights f
          join airports_data a on a.airport_code=f.departure_airport
          join airports_data a_ar on a_ar.airport_code=f.arrival_airport
    where upper(a.city)='ST. PETERSBURG'
            and upper(a_ar.city)='MOSCOW'
            and f.date_departure > to_date('01.09.2017 07:55:00', 'dd.mm.yyyy hh24:mi:ss')
    order by f.date_departure) t
where rownum<=5
```

Вывести 10 бронирований с самой высокой стоимостью

```
select *
from (select   *
        from  bookings
        order by total_amount desc)
where rownum<10
```

Группировка и множественные операции

GROUP BY Запросы с группировкой

Использование фразы GROUP BY позволяет сгруппировать строки в группы, имеющие одинаковые значения указанного поля:

aircraft_code	ORDER BY aircraft_code		GROUP BY aircraft_code
319	319		319
321=	321	=	321
733	321		733
321	733		

К группам, полученным после применения GROUP BY, можно применить любую из стандартных агрегатных функций.

Основные агрегатные функции

- AVG() - функция возвращает среднее значение числового столбца.
- COUNT() - данная функция возвращает количество строк, которые соответствует определенным критериям.
- FIRST() - данная функция возвращает первое значение для выбранного столбца.
- LAST() - данная функция возвращает последнее значение для выбранного столбца.
- MAX() - функция возвращает наибольшее значение для выбранного столбца.
- MIN() - функция возвращает наименьшее значение для выбранного столбца.
- SUM() - функция возвращает сумму числового столбца.

GROUP BY Использование агрегатных функций

Список самолетов и количество мест в них:

```
select aircraft_code
      , count(*)
from bookings.seats
group by aircraft_code
```

Список самолетов и количество мест в них с учетом класса места

```
select aircraft_code
      , fare_conditions
      , count(*)
from bookings.seats
group by aircraft_code, fare_conditions
order by 1,2
```

Вывести количество рейсов в сентябре 2017г. по дням

```
select trunc(t.date_departure) date_departure
        , count(*) countFlight
from lanit.flights t
where trunc(t.date_departure, 'mm')=to_date('01.09.2017','dd.mm.yyyy')
group by trunc(t.date_departure)
order by 1
```

Вывести количество рейсов в сентябре 2017г. по дням и аэропортам

```
select a.airport_name
          , trunc(f.date_departure)
          , count(*) countairport
from lanit.flights f
        , airports_data a
where trunc(f.date_departure,'mm')='01.09.2017'
           and f.departure_airport=a.airport_code
group by trunc(f.date_departure), a.airport_name
order by 1,2
```

GROUP BY Условия поиска групп. Предложение HAVING

Предложение HAVING, используемое совместно с GROUP BY, позволяет исключить из результата группы, неудовлетворяющие условию (так же, как WHERE позволяет исключить строки).

Получить кол-во мест в самолете по каждому типу, где мест больше 20

```
select aircraft_code
      , fare_conditions
      , count(*)
from bookings.seats
group by aircraft_code, fare_conditions
having count(*) >=20
order by 1,2
```

Вывести дни в сентябре 2017 с количество рейсов больше 500.

```
select trunc(t.date_departure) date_departure
        , count(*) countFlight
from lanit.flights t
where trunc(t.date_departure, 'mm')=to_date('01.09.2017','dd.mm.yyyy')
group by trunc(t.date_departure)
having count(*)>500
order by 1
```

Вывести список городов, в которых несколько аэропортов

```
select a.city
from airports_data a
group by a.city
having   count(*) > 1
```

Вложенные и коррелируемые подзапросы

Вложенным запросом (подзапросом) называется запрос, содержащийся в предложении WHERE или HAVING другого оператора SQL.

Вложенный подзапрос возвращает одно значение

```
SELECT список_полей FROM имя_табл1
WHERE имя_поля1 = (SELECT имя_поля2 FROM имя_табл2 WHERE условие)
```

Вложенный подзапрос возвращает несколько значений

```
SELECT список_полей FROM имя_табл1
WHERE имя_поля1 IN (SELECT имя_поля2 FROM имя_табл2 WHERE условие)
```

Коррелируемым подзапросом называется подзапрос, который содержит ссылку на столбцы таблицы внешнего запроса.

```
SELECT список_полей FROM имя_табл1
WHERE EXIST (SELECT имя_поля2 FROM имя_табл2
        WHERE имя_табл1.поле= имя_табл2.поле)
```

Получить список мест в самолетах с названием на «BO»

```
select s.aircraft_code, seat_no, fare_conditions
from seats s
where s.aircraft_code in (select a.aircraft_code
                          from aircrafts_data a
                          where upper(a.model) like 'BO%')
order by s.aircraft_code, seat_no, fare_conditions
```

! Перепишите данный запрос используя простое соединение таблиц.

Как правило в городе есть только один аэропорт, исключение:

```
select a.airport_code as code, a.airport_name, a.city
from airports_data a
where a.city in (select aa.city
                 from airports_data aa
                 group by aa.city
                 having   count(*) > 1
order by a.city, a.airport_code;
```

Показать все рейсы 01.09.2017 по самолетам с дальностью полета свыше 8тыс. км

```
select *
from lanit.flights t
where trunc(t.date_departure)=to_date('01.09.2017','dd.mm.yyyy')
        and (select a.range
                from aircrafts_data a
                where a.aircraft_code=t.aircraft_code)>8000
order by t.date_departure
```

Квантор существования EXISTS

В языке SQL предикат с квантором существования представляется выражением вида:

[NOT] EXISTS (SELECT…FROM…WHERE…),

которое следует за фразой WHERE. Такое выражение считается истинным, если подзапрос возвращает непустое множество (существует хотя бы 1 строка, которую возвращает подзапрос). На практике подзапрос всегда будет коррелированным.

Показать данные по самолетам, у которых есть рейсы 01.09.2017

```
select a.*
from aircrafts_data a
where exists (select 1 from flights f
                    where f.aircraft_code=a.aircraft_code
                          and trunc(f.date_departure)='01.09.2017')
```

! Показать данные по самолетам, у которых нет рейсов 01.09.2017

! Показать данные по самолетам, у которых есть рейсы 01.09.2017 и в которых больше 150 мест

Многократное сравнение ANY и ALL

Синтаксис многократного сравнения: **проверяемое_выражение = | <> | < | <= | > | >= ANY | ALL вложенный_запрос**

Получить список перелетов, которые дороже всех перелетов по билету 0005432661915

```
select tf.ticket_no, tf.flight_id, tf.amount
from ticket_flights tf
        join flights f on tf.flight_id = f.flight_id
where trunc(f.date_departure)='01.09.17'
          and tf.ticket_no <> '0005432661915'
          and tf.amount > all (select t_tf.amount
                                from ticket_flights t_tf
                               where t_tf.ticket_no='0005432661915')
```

Найти перелеты стоимость которых равна стоимости любого перелета по билету 0005432661915

```
select tf.ticket_no, tf.flight_id, tf.amount
from ticket_flights tf
        join flights f on tf.flight_id = f.flight_id
where trunc(f.date_departure)='01.09.17'
          and tf.ticket_no <> '0005432661915'
          and tf.amount = any (select t_tf.amount  from ticket_flights t_tf
                                 where t_tf.ticket_no='0005432661915')
```

Реляционные операторы

Теоретико-множественные операторы:

- Декартово произведение Cross join
- Объединение Union
- Пересечение Intersect
- Разность Minus

Специальные реляционные операторы:

- Выборка where
- Соединение inner join, outer join
- Существование exists

Оператор CASE

Оператор **CASE** имеет функциональность IF-THEN-ELSE и используется в SQL предложении

```
CASE WHEN condition_1 THEN result_1
     WHEN condition_2 THEN result_2
     ...
     WHEN condition_n THEN result_n
     ELSE result
END
```

Вывести данные по билетам за август 2017 (номер билета, ФИО пассажира, email или телефон)

```
select t.ticket_no
     , t.passenger_name
     , case when t.email is not null then t.email
            else t.phone
       end contact
from bookings b
     join tickets t on b.book_ref=t.book_ref
where trunc(b.book_date,'mm')='01.08.17'
```

Функция DECODE. Имеет функциональные возможности оператора IF-THEN-ELSE.

DECODE(выражение, значение 1, результат 1 [, значение N , результат N]… [, иначе результат])

Вывести список аэропортов по условию

```
select decode(upper(city)
                    , 'MOSCOW',  city||' '||airport_name
                    , 'NALCHIK',  city
                    , airport_name
                   ) name
      , city
      , airport_name
from lanit.airports_data;
```

Функция NVL. позволяет заменить значение, когда встречается Null значение.

NVL(строка, значение)

Вывести данные по билетам за август 2017 (номер билета, ФИО пассажира, email или телефон)

```
select t.ticket_no
        , t.passenger_name
        , nvl(t.email, t.phone)
          end contact
from bookings b
       join tickets t on b.book_ref=t.book_ref
where trunc(b.book_date,'mm')='01.08.17'
```

Функция LISTAGG

Объединяет значения «выражение» для каждой группы

LISTAGG (выражение[, ‘разделитель’]) **WITHIN GROUP** (order_by колонка1..)

Вывести данные о посадочных местах в самолетах

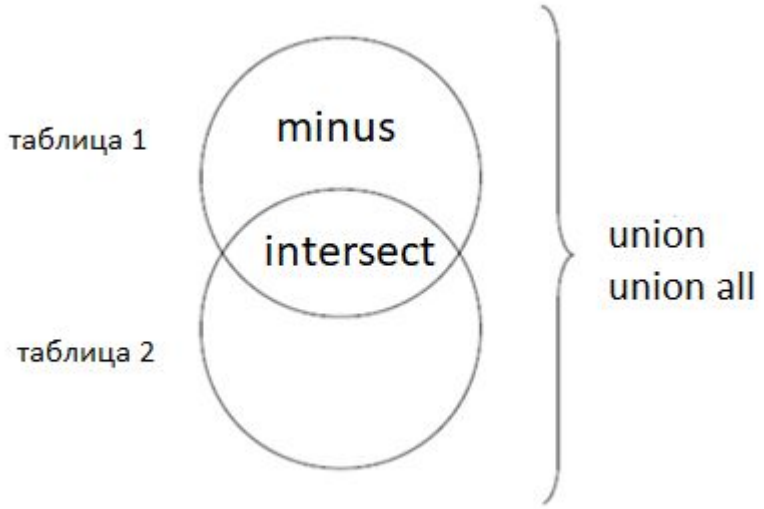
```
select s2.aircraft_code
      , listagg (s2.fare_conditions ||'('||s2.num||')', ', ') within group (order by fare_conditions) fare_conditions
from (
        select s.aircraft_code, s.fare_conditions
              , count(*) as num
        from seats s
        group by s.aircraft_code, s.fare_conditions
       ) s2
group by s2.aircraft_code
order by s2.aircraft_code
```

AIRCRAFT_CODE	FARE_CONDITIONS
CN1	Economy(12)
CR2	Economy(50)
SU9	Business(12), Economy(85)
319	Business(20), Economy(96)
320	Business(20), Economy(120)
321	Business(28), Economy(142)
733	Business(12), Economy(118)
763	Business(30), Economy(192)
773	Business(30), Comfort(48), Economy(324)

CTE (Common Table Expressions) Общие табличные выражения

```
with listdata as
   (select ad.aircraft_code
         , ad.model
         , sum(case when to_char(f.date_departure,'mm')=1 then 1 else 0 end) count1
         , sum(case when to_char(f.date_departure,'mm')=2 then 1 else 0 end) count2
         , sum(case when to_char(f.date_departure,'mm')=3 then 1 else 0 end) count3
   from aircrafts_data ad
         , flights f
   where ad.aircraft_code=f.aircraft_code
           and trunc(f.date_departure) between '01.01.17‘ and ’31.03.17’
   group by ad.model, ad.aircraft_code
)
 select aircraft_code, model, count1, count2, count3  from listdata
 union all
 select null, 'Минимум', min(count1), min(count2), min(count3)  from listdata
 union all
 select null, ‘Максимум', max(count1), max(count2), max(count3)  from listdata
```

Операторы объединения запросов



UNION возвращает все строки из обоих операторов SELECT; повторяющиеся значения удаляются.

UNION ALL возвращает все строки из обоих операторов SELECT; повторяющиеся значения показываются.

INTERSECT возвращает строки, которые возвращены и первым, и вторым оператором SELECT.

MINUS возвращает строки, которые возвращены первым оператором SELECT, исключая те, которые возвращены вторым оператором.

Количество и порядок столбцов, возвращаемых SELECT из обеих таблиц, должны совпадать.

Пример объединения UNION, UNION ALL

aircrafts1

MODEL
Airbus A320-200
Airbus A321-200
Boeing 737-300
Bombardier CRJ-200

aircrafts2

MODEL
Airbus A319-100
Airbus A320-200
Airbus A321-200
Boeing 767-300
Boeing 777-300
Cessna 208 Caravan
Sukhoi Superjet-100

```
select model from aircrafts1
union
select model from aircrafts2
order by 1
```

MODEL
Airbus A319-100
Airbus A320-200
Airbus A321-200
Boeing 737-300
Boeing 767-300
Boeing 777-300
Bombardier CRJ-200
Cessna 208 Caravan
Sukhoi Superjet-100

```
select model from aircrafts1
union all
select model from aircrafts2
order by 1
```

MODEL
Airbus A319-100
Airbus A320-200
Airbus A320-200
Airbus A321-200
Airbus A321-200
Boeing 737-300
Boeing 767-300
Boeing 777-300
Bombardier CRJ-200
Cessna 208 Caravan
Sukhoi Superjet-100

Операторы соединения MINUS, INTERSECT

aircrafts1

MODEL
Airbus A320-200
Airbus A321-200
Boeing 737-300
Bombardier CRJ-200

aircrafts2

MODEL
Airbus A319-100
Airbus A320-200
Airbus A321-200
Boeing 767-300
Boeing 777-300
Cessna 208 Caravan
Sukhoi Superjet-100

```
select model from aircrafts1
intersect
select model from aircrafts2
```

MODEL
Airbus A320-200
Airbus A321-200

```
select model from aircrafts1
minus
select model from aircrafts2
```

MODEL
Boeing 737-300
Bombardier CRJ-200

Оператор CROSS JOIN (декартово произведение)

Порядок таблиц для оператора неважен

Оператор SQL CROSS JOIN формирует таблицу перекрестным соединением (декартовым произведением) двух таблиц. При использовании оператора SQL CROSS JOIN каждая строка левой таблицы сцепляется с каждой строкой правой таблицы. В результате получается таблица со всеми возможными сочетаниями строк обеих таблиц

```
SELECT *
FROM city c
        CROSS JOIN person p
```

city (города)

ID_CITY	NAME
1	City1
2	City2
3	City3

person (люди)

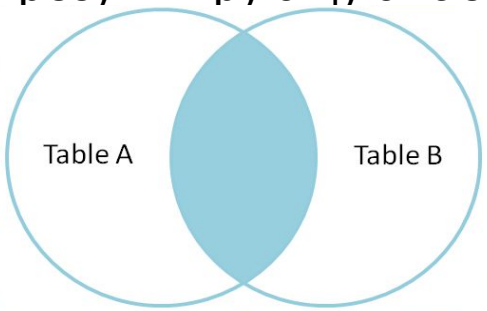
ID_PERSON	NAME	ID_CITY
1	Human1	1
2	Human2	2
3	Human3	1
4	Human4	4

Результирующая таблица

CITY.ID_CITY	NAME	PERSONS.ID_CITY	ID_PERSO	NAME
1	City1	1	1	Human1
1	City1	2	2	Human2
1	City1	1	3	Human3
1	City1	4	4	Human4
2	City2	1	1	Human1
2	City2	2	2	Human2
2	City2	1	3	Human3
2	City2	4	4	Human4
3	City3	1	1	Human1
3	City3	2	2	Human2
3	City3	1	3	Human3
3	City3	4	4	Human4

Оператор INNER JOIN (внутреннее соединение)

Оператор *INNER JOIN* соединяет две таблицы. **Порядок таблиц для оператора неважен**. Каждая строка из первой (левой) таблицы, сопоставляется с каждой строкой из второй (правой) таблицы, после чего происходит проверка условия. Если условие истинно, то строки попадают в результирующую таблицу.




```
SELECT *
FROM city c
        INNER JOIN person p ON p.id_city = c.id_city
```

city (города)

ID_CITY	NAME
1	City1
2	City2
3	City3

person (люди)

ID_PERSON	NAME	ID_CITY
1	Human1	1
2	Human2	2
3	Human3	1
4	Human4	4

Промежуточная таблица

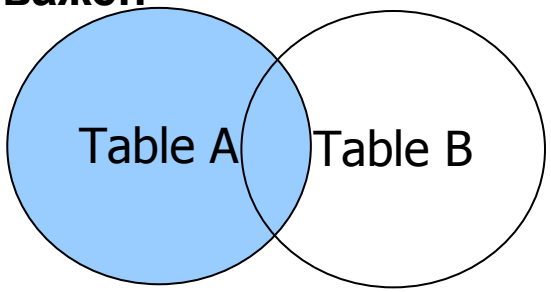
CITY.ID_CITY	NAME	PERSONS.ID_CITY	ID_PERSO	NAME
1	**City1**	**1**	**1**	**Human1**
1	City1	2	2	Human2
1	**City1**	**1**	**3**	**Human3**
1	City1	4	4	Human4
2	City2	1	1	Human1
2	**City2**	**2**	**2**	**Human2**
2	City2	1	3	Human3
2	City2	4	4	Human4
3	City3	1	1	Human1
3	City3	2	2	Human2
3	City3	1	3	Human3
3	City3	4	4	Human4

Результирующая таблица

CITY.ID_CITY	NAME	PERSONS.ID_CITY	ID_PERSO	NAME
1	City1	1	1	Human1
1	City1	1	3	Human3
2	City2	2	2	Human2

Оператор LEFT OUTER JOIN (левое внешнее соединение)

Порядок таблиц для оператора важен




```
SELECT *
FROM city c
        LEFT OUTER JOIN person p ON p.id_city = c.id_city
```

city (города)

ID_CITY	NAME
1	City1
2	City2
3	City3

person (люди)

ID_PERSON	NAME	ID_CITY
1	Human1	1
2	Human2	2
3	Human3	1
4	Human4	4

1. Сначала происходит формирование таблицы внутренним соединением левой и правой таблиц

CITY.ID_CITY	NAME	PERSONS.ID_CITY	ID_PERSO	NAME
1	**City1**	**1**	**1**	**Human1**
1	City1	2	2	Human2
1	**City1**	**1**	**3**	**Human3**
1	City1	4	4	Human4
2	City2	1	1	Human1
2	**City2**	**2**	**2**	**Human2**
2	City2	1	3	Human3
2	City2	4	4	Human4
3	City3	1	1	Human1
3	City3	2	2	Human2
3	City3	1	3	Human3
3	City3	4	4	Human4

2. Затем, в результат добавляются записи левой таблицы не вошедшие в результат формирования таблицы внутренним соединением. Для них, соответствующие записи из правой таблицы

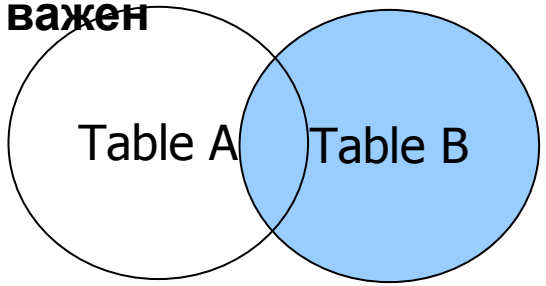
3	City3	NULL	NULL	NULL

Результирующая таблица

CITY.ID_CITY	NAME	PERSONS.ID_CITY	ID_PERSO	NAME
1	City1	1	1	Human1
1	City1	1	3	Human3
2	City2	2	2	Human2
3	City3	NULL	NULL	NULL

Оператор RIGHT OUTER JOIN (правое внешнее соединения)

Порядок таблиц для оператора важен




```
SELECT *
FROM city c
        RIGHT OUTER JOIN person p ON p.id_city = c.id_city
```

city (города)

ID_CITY	NAME
1	City1
2	City2
3	City3

person (люди)

ID_PERSON	NAME	ID_CITY
1	Human1	1
2	Human2	2
3	Human3	1
4	Human4	4

Результирующая таблица

CITY.ID_CITY	NAME	PERSONS.ID_CITY	ID_PERSO	NAME
1	City1	1	1	Human1
1	City1	1	3	Human3
2	City2	2	2	Human2
NULL	NULL	4	4	Human4

1. Сначала происходит формирование таблицы внутренним соединением левой и правой таблиц

CITY.ID_CITY	NAME	PERSONS.ID_CITY	ID_PERSO	NAME
1	**City1**	**1**	**1**	**Human1**
1	City1	2	2	Human2
1	**City1**	**1**	**3**	**Human3**
1	City1	4	4	Human4
2	City2	1	1	Human1
2	**City2**	**2**	**2**	**Human2**
2	City2	1	3	Human3
2	City2	4	4	Human4
3	City3	1	1	Human1
3	City3	2	2	Human2
3	City3	1	3	Human3
3	City3	4	4	Human4

2. Затем, в результат добавляются записи правой таблицы не вошедшие в результат формирования таблицы внутренним соединением. Для них, соответствующие записи из левой таблицы заполняются значениями NULL

NULL	NULL	4	4	Human4

Оператор FULL OUTER JOIN (полное внешнее соединение)

Порядок таблиц для оператора неважен


```
SELECT *
FROM city c
        FULL OUTER JOIN person p ON p.id_city = c.id_city
```

city (города)

ID_CITY	NAME
1	City1
2	City2
3	City3

person (люди)

ID_PERSON	NAME	ID_CITY
1	Human1	1
2	Human2	2
3	Human3	1
4	Human4	4

Результирующая таблица

CITY.ID_CITY	NAME	PERSONS.ID_CITY	ID_PERSO	NAME
1	City1	1	1	Human1
1	City1	1	3	Human3
2	City2	2	2	Human2
3	City3	NULL	NULL	NULL
NULL	NULL	4	4	Human4

1. Сначала происходит формирование таблицы внутренним соединением левой и правой таблиц

CITY.ID_CITY	NAME	PERSONS.ID_CITY	ID_PERSO	NAME
1	**City1**	**1**	**1**	**Human1**
1	City1	2	2	Human2
1	**City1**	**1**	**3**	**Human3**
1	City1	4	4	Human4
2	City2	1	1	Human1
2	**City2**	**2**	**2**	**Human2**
2	City2	1	3	Human3
2	City2	4	4	Human4
3	City3	1	1	Human1
3	City3	2	2	Human2
3	City3	1	3	Human3
3	City3	4	4	Human4

2. Затем, в результат добавляются записи левой таблицы не вошедшие в результат формирования таблицы внутренним соединением. Для них, соответствующие записи из правой таблицы

3	City3	NULL	NULL	NULL

3. Затем, в результат добавляются записи правой таблицы не вошедшие в результат формирования таблицы внутренним соединением. Для них, соответствующие записи из левой таблицы

NULL	NULL	4	4	Human4

Вывести все перелеты по билету номер 0005432661915

```
select   to_char(f.date_departure, 'dd.mm.yyyy') date_departure
           , a.city||' ('||a.airport_code||')' departure
           , a_ar.city||' ('||a_ar.airport_code||')' departure
           , tf.fare_conditions
           , tf.amount
from ticket_flights tf
       join flights f on tf.flight_id = f.flight_id
       join airports_data a on a.airport_code=f.departure_airport
       join airports_data a_ar on a_ar.airport_code=f.arrival_airport
where    tf.ticket_no = '0005432661915'
order by f.scheduled_departure
```

! добавить информацию: название самолета, место и данные пассажира

SAMPLE – выборка случайных строк из таблицы

SAMPLE (процент) [SEED (значение)]

Процент может принимать значение от 0.000001 до 100 (не включая)

Если указать параметр SEED, то БД будет попытаться вернуть одну и туже выборку, иначе каждый раз будет возвращаться разные строки таблицы.

Выбрать 10% данных из таблицы билетов

```
select t.*
from tickets sample(10) t;
```

Выбрать 10% данных из таблицы билетов с именем пассажира DMITRIY SERGEEV

```
select *
from tickets sample(10) t
where t.passenger_name ='DMITRIY SERGEEV';
```

Выборка одной записи

```
select t.*
from tickets sample(10) t
where t.passenger_name='DMITRIY SERGEEV'
        and rownum=1;
```

При выполнении запроса в первую очередь выполняется sample – будет выбрано 10% записей, после этого выполнится where. При выборе 10% записей, в списке может не оказаться пассажиров с таким именем.

Выборка N первых записей. Выбрать 10 самых дорогих рейсов

Способ 1. С помощью конструкции "SELECT FROM SELECT“

```
select *
from
   (select t.ticket_no, tf.flight_id, t.passenger_name
       , tf.amount
    from tickets t
            join ticket_flights tf on t.ticket_no=tf.ticket_no
    order by tf.amount desc)
where rownum<=10
```

Способ 2. "Классика" с Oracle 7

```
select *
from
   (select t.ticket_no, tf.flight_id, t.passenger_name
       , tf.amount
            , row_number() over (order by tf.amount desc) rn
    from tickets t
       join ticket_flights tf on t.ticket_no=tf.ticket_no
   )
where rn<=10
```

Способ 3. "Новомодный" с Oracle 12c.

```
select t.ticket_no, tf.flight_id, t.passenger_name
          , tf.amount
from tickets t
        join ticket_flights tf on t.ticket_no=tf.ticket_no
order by tf.amount desc
fetch first 10 rows only
-- или одна строка fetch first row only
```

fetch first 10 rows with ties (если надо вернуть все записи с одинаковым значением)

```
select t.ticket_no, tf.flight_id, t.passenger_name, tf.amount
from tickets t
        join ticket_flights tf on t.ticket_no=tf.ticket_no
order by tf.amount desc
fetch first 10 rows with ties
```

offset m rows fetch next n rows only (выбор n строк после m записей)

```
select t.ticket_no, tf.flight_id, t.passenger_name, tf.amount
from tickets t
        join ticket_flights tf on t.ticket_no=tf.ticket_no
order by tf.amount desc
offset 25 rows fetch next 10 rows only
```

fetch first 1 percent rows only (выбор 1% записей)

```
select t.ticket_no, tf.flight_id, t.passenger_name, tf.amount
from tickets t
    join ticket_flights tf on t.ticket_no=tf.ticket_no
order by tf.amount desc
fetch first 1 percent rows only
```

Запросы модификации и определения данных

Операции модификации данных (DML) INSERT

Для модификации данных используются три оператора: INSERT, DELETE и UPDATE.

Оператор **INSERT** используется для вставки одной записи или несколько записей в таблицу
Синтаксис оператора INSERT при вставке одной записи с помощью ключевого слова VALUES:

```
INSERT INTO table (column1, column2, … column_n )
VALUES (expression1, expression2, … expression_n );
```

Добавить новый самолет в БД:

```
INSERT INTO aircrafts_data (aircraft_code, range, model)
VALUES ('як-42', 12000, 'яковлев як-42');
```

Многострочный оператор INSERT добавляет в таблицу несколько строк:

```
INSERT INTO table (column1, column2, … column_n )
SELECT expression1, expression2, … expression_n
FROM source_table
[WHERE conditions]
```

! Добавить архивные модели самолетов (aircrafts_data_archive)
! Вставить недостающие данные о само
летах с дальностью полета менее 5тыс. км. из архива (aircrafts_data_archive) в aircrafts_data

Операции модификации данных (DML) DELETE

Удаление строк из таблицы БД осуществляется с помощью оператора DELETE (удалить):

DELETE FROM имя_таблицы
[**WHERE** условие_поиска],

где условие поиска может быть вложенным запросом.

Удалить данные из архива самолетов.

DELETE FROM aircrafts_data_archive **WHERE** aircraft_code='';

Удалить все билеты по бронированию номер

Примечание. Отсутствие предложения WHERE приводит к удалению ВСЕХ строк из указанной таблицы.

Операции модификации данных (DML) UPDATE

Обновление значения одного или нескольких столбцов в выбранных строках одной таблицы БД осуществляется с помощью оператора UPDATE (обновить):

UPDATE имя_таблицы
  SET имя_столбца1 = выражение1
    , имя_столбца2 = выражение2…
[**WHERE** условие_поиска]

Увеличить на 20% дальность полета самолетов «Boeing».

```
update aircrafts_data
set range=1.2*range
where upper(model) like 'BO%'
```

Примечание. Отсутствие предложения WHERE приводит к обновлению ВСЕХ строк из указанной таблицы.

Операции определения данных (DDL)

Команды DDL:

CREATE – создает объект БД;

ALTER – изменяет определение существующего объекта;

DROP – удаляет ранее созданный объект.

CREATE TABLE – оператор позволяет создавать и определять таблицу

CREATE TABLE table_name
(
  column1 datatype [**NULL | NOT NULL],**
  column2 datatype [**NULL | NOT NULL],**
  …
  column_n datatype [**NULL | NOT NULL]**
[[**CONSTRAINT** <имя_ограничения>] <ограничение уровня колонки>]…
[[**CONSTRAINT** <имя_ограничения>] <ограничение уровня таблицы>]
)

Ограничения

PRIMARY KEY – определение первичного ключа таблицы;

UNIQUE – обеспечение уникальности значений в колонке;

NULL / NOT NULL – разрешение или запрещение неопределенных значений в колонке;

CHECK <условие> – задание условия на значение данных в колонке;

[FOREIGN KEY <имя_колонки>] REFERENCES <имя_таблицы> <имя_колонки> – определение внешнего ключа для таблицы.

Создания таблицы AIRPORTS_DATA

```
CREATE TABLE AIRPORTS_DATA
  (
   AIRCRAFT_CODE CHAR(3) NOT NULL ENABLE,
   RANGE NUMBER(10,0) NOT NULL ENABLE,
   MODEL VARCHAR2(100) NOT NULL ENABLE,
   CONSTRAINT AIRCRAFTS_PKEY PRIMARY KEY (AIRCRAFT_CODE)
  )
```

CREATE TABLE AS ОПЕРАТОР – создание таблицы на основе существующей, путем копирования столбцов существующей таблицы

```
CREATE TABLE new_table AS (SELECT * FROM old_table);
```

```
CREATE TABLE new_table AS (SELECT t1.col1, t2.col2, t2.col6
                           FROM old_table1 t1
                             join old_table2 t2 on t1.col1=t2.col1
                    where t1.col5<1000);
```

ALTER TABLE – оператор который позволяет добавить, изменить, удалить, переименовать столбец или переименовать таблицу

Добавить столбец в таблицу

```
ALTER TABLE table_name
        ADD column_name column-definition;
```

Изменить столбец в таблице

```
ALTER TABLE table_name
        MODIFY column_name column_type;
```

Удаление столбца из таблицы

```
ALTER TABLE table_name
        DROP COLUMN column_name;
```

Переименование столбца в таблице

```
ALTER TABLE table_name
        RENAME COLUMN old_name to new_name;
```

Переименовать таблицу

```
ALTER TABLE table_name
        RENAME TO new_table_name;
```

Добавить ограничение

```
ALTER TABLE table_name
        ADD CONSTRAINT name
        PRIMARY KEY (table_id);
```

Удалить ограничение

```
ALTER TABLE table_name
        DROP CONSTRAINT name
```

DROP TABLE – оператор позволяет очистить или удалить таблицу из БД

DROP TABLE [schema_name].table_name
[CASCADE CONSTRAINTS]
[PURGE];

CASCADE CONSTRAINTS - необязательный. Если этот параметр задан, все ограничения ссылочной целостности будут также удалены.

PURGE - необязательный. Если указано, таблица и ее зависимые объекты будут удалены из корзины, и вы не сможете восстановить таблицу. Если PURGE не указан, таблица и ее зависимые объекты помещаются в мусорную корзину и могут быть восстановлены позже, если это необходимо

Представления, привилегии, хранимые процедуры

VIEW - представление

Представление – представляет собой виртуальную таблицу, которая физически не существует. Она создается с помощью запроса соединяющего одну или несколько таблиц. Обратиться к представлению можно, как к обычной таблице.

Создание представления

```
CREATE VIEW view_name AS
    SELECT columns
    FROM tables
    [WHERE conditions];
```

Изменение представления

```
CREATE OR REPLACE VIEW view_name AS
    SELECT columns
    FROM tables
    [WHERE conditions];
```

Удаление представления

```
DROP VIEW view_name
```

Выбор данных из представления

```
SELECT * FROM view_name
```

Модификация данных через представления

Представление модифицируемое, если относительно него можно использовать все три команды – INSERT, UPDATE, DELETE, и оно создано на основе одной таблицы БД.

Условия модифицируемости представлений:

представление должно формироваться из одной базовой таблицы;

оно не должно содержать ключевого слова DISTINCT в списке фразы SELECT;

- список фразы SELECT должен содержать только имена столбцов;
- представление не должно содержать предложений GROUP BY или HAVING, и подзапросов в предложениях FROM и WHERE;
- для операторов INSERT и UPDATE представление должно содержать все столбцы базовой таблицы, которые имеют ограничения NOT NULL.

Привилегии. Директивы GRANT и REVOKE

GRANT (допуск)
REVOKE (отмена)

Привилегии для таблиц и представлений:

- SELECT – позволяет считывать данные
- INSERT – позволяет вставлять новые записи
- UPDATE – позволяет модифицировать записи
- DELETE – позволяет удалять записи

Дополнительны привилегии в СУБД

- ALTER – позволяет модифицировать структуру таблиц (DB2, Oracle)
- EXECUTE – позволяет выполнять хранимые процедуры

Привилегии. Синтаксис GRANT

GRANT {**SELECT**|**INSERT**|**DELETE**|(**UPDATE** столбец, …)},…
ON таблица **TO** {пользователь | **PUBLIC**} [**WITH GRANT OPTION**]

Предоставить пользователю Ivanov полномочия для осуществления выборки и модификации фамилий в таблице Students с правом предоставления полномочий.

GRANT SELECT, UPDATE StName **ON** Students **TO** Ivanov **WITH GRANT OPTION**

Привилегии. Синтаксис REVOKE

REVOKE {{**SELECT | INSERT | DELETE | UPDATE**},…|**ALL PRIVILEGES**}
ON таблица,… **FROM** {пользователь | **PUBLIC**},… {**CASCADE** | **RESTRICT**}

Снять с пользователя Ivanov полномочия для осуществления модификации фамилий в таблице Students. Также снять эту привилегию со всех пользователей, которым она была предоставлена Ивановым.

REVOKE UPDATE ON Students **FROM** Ivanov **CASCADE**

Хранимые процедуры

Хранимая процедура (Stored Procedure, SP) – набор заранее скомпилированных операторов SQL и операторов управления программой, который хранится как объект БД.

SP расширяют стандартные возможности SQL, позволяя использовать входные и выходные параметры, операторы принятия решения и объявления переменных. Обеспечивают значительный выигрыш в быстродействии, поскольку операторы SQL, содержащиеся в SP, заранее скомпилированы.

SP используются во всех случаях, когда необходимо получить максимальное быстродействие и свести код SQL в единую программу. Чаще всего используются SP, выполняющие вставку, удаление и обновление данных, а также формирующие данные для отчетов.
SP используются в качестве **механизма защиты**: если нельзя предоставлять прямой доступ пользователям к таблицам и представлениям БД, тогда им предоставляется доступ к таблицам только по чтению, а для выполнения таких операций, как UPDATE и DELETE, создаются соответствующие SP, на выполнение которых пользователи и получают права, т.е. пользователи получают доступ к таблицам БД только путем выполнения SP.

Аналитические функции

Применяются для подсчета промежуточных итогов, процентов по группе, среднего, ранжирование запросов и т.д.

ИМЯ_ФУНКЦИИ(<аргумент>,< аргумент >, . . .)

OVER <конструкция_фрагментации(группы)> <конструкция_упорядочения> <конструкция_окна>)

OVER — ключ. слово, идентифицирующее эту функцию как аналитическую. Конструкция после ключевого слова OVER описывает срез данных, "по которому" будет вычисляться функция.

<конструкция_фрагментации> - **PARTITION BY столбец** - необязательная конструкция. Если конструкция фрагментации не задана, все результирующее множество считается одним большим фрагментом.

<конструкция_упорядочения> - **ORDER BY столбец** - необязательная конструкция. Задает критерий сортировки данных в каждой группе. Синтаксис ORDER BY выражение [ASC | DESC] [NULLS FIRST | NULLS LAST]

Вывести все бронирования за август 2017, показать общую сумму и количество

```
select to_char(b.book_date,'dd.mm.yyyy') dates
        , b.total_amount
        , sum(b.total_amount) over () sumall
        , count(b.total_amount) over () countall
from bookings b
where trunc(b.book_date,'mm')='01.08.2017'
order by b.book_date
```

DATES	TOTAL_AMOUNT	SUMALL	COUNTALL
01.08.2017	12,400	6,993,792,100	88,647
01.08.2017	226,200	6,993,792,100	88,647
01.08.2017	29,400	6,993,792,100	88,647
01.08.2017	23,400	6,993,792,100	88,647
01.08.2017	42,800	6,993,792,100	88,647
01.08.2017	71,600	6,993,792,100	88,647
01.08.2017	60,400	6,993,792,100	88,647
01.08.2017	186,900	6,993,792,100	88,647
01.08.2017	95,500	6,993,792,100	88,647
01.08.2017	26,000	6,993,792,100	88,647
01.08.2017	66,600	6,993,792,100	88,647
01.08.2017	95,700	6,993,792,100	88,647

Вывести все бронирования за август 2017, показать общую сумму и количество по дню

```
select to_char(b.book_date,'dd.mm.yyyy') dates
        , b.total_amount
        , sum(b.total_amount) over (partition by trunc(b.book_date)) sumday
        , max(b.total_amount) over (partition by trunc(b.book_date)) maxday
        , count(b.total_amount) over (partition by trunc(b.book_date)) countday
        , round(b.total_amount
                /sum(b.total_amount) over (partition by trunc(b.book_date))
               *100,2) percent
from bookings b
where trunc(b.book_date,'mm')='01.08.17'
order by b.book_date
```

DATES	TOTAL_AMOUNT	SUMDAY	MAXDAY	COUNTDAY	PERCENT
01.08.2017	226,200	450,124,400	805,800	5,554	0.05
01.08.2017	12,400	450,124,400	805,800	5,554	0
01.08.2017	23,400	450,124,400	805,800	5,554	0.01
01.08.2017	29,400	450,124,400	805,800	5,554	0.01
01.08.2017	71,600	450,124,400	805,800	5,554	0.02
01.08.2017	42,800	450,124,400	805,800	5,554	0.01
01.08.2017	60,400	450,124,400	805,800	5,554	0.01
01.08.2017	95,500	450,124,400	805,800	5,554	0.02
01.08.2017	186,900	450,124,400	805,800	5,554	0.04
01.08.2017	13,600	450,124,400	805,800	5,554	0
01.08.2017	120,000	450,124,400	805,800	5,554	0.03
01.08.2017	95,700	450,124,400	805,800	5,554	0.02
01.08.2017	66,600	450,124,400	805,800	5,554	0.01
01.08.2017	26,000	450,124,400	805,800	5,554	0.01

Транзакции (Неделимая последовательность)

Транзакцией в SQL называется логически неделимая последовательность операторов, рассматриваемая как единое целое.

Результаты выполнения операторов, входящих в транзакцию, могут быть либо сохранены в БД при помощи оператора **COMMIT**, либо полностью аннулированы оператором **ROLLBACK** (или **ROLLBACK** до точки сохранения).

Транзакция начинается с 1-го выполняемого оператора, либо с 1-го оператора после **COMMIT** или ROLLBACK.

Транзакция заканчивается при выполнении операторов **COMMIT** или **ROLLBACK**.

Чтобы оградить данные от модифицирования другими пользователями, в начале транзакции следует выполнить оператор SET TRANSACTION READ ONLY. При этом не допускается и изменение данным самим пользователем, издавшим директиву.

Древовидный запрос(предложение CONNECT BY)

Древовидным запросом называется запрос, в котором присутствует предложение CONNECT BY, предназначенное для отображения строк результата в определенном иерархическом порядке.

Начиная с корня, описанного предложением STARTWITH, запрос просматривает каждую соединенную с корнем ветвь.

Задание 1 (ответы см. в заметках к слайду)

1. Вывести список самолетов с кодами 320, 321, 733;
2. Вывести список самолетов с кодом не на 3;
3. Найти билеты оформленные на имя «OLGA», с емайлом «OLGA» или без емайла;
4. Найти самолеты с дальность полета 5600, 5700. Отсортировать список по убыванию дальности полета;
5. Найти аэропорты в Moscow. Вывести название аэропорта вместе с городом. Отсортировать по полученному названию;
6. Вывести список всех городов без повторов в зоне «Europe»;
7. Найти бронирование с кодом на «3A4» и вывести сумму брони со скидкой 10%
8. Вывести все данные по местам в самолете с кодом 320 и классом «Business»в формате «Данные по месту: номер места»
9. Сайт http://www.sql-ex.ru/learn_exercises.php. Задания: 1-5
10. Архитектура базы данных Oracle.pdf (стр. 1 – 43)

Задание 2 (ответы см. в заметках к слайду)

1. Найти максимальную и минимальную сумму бронирования в 2017 году;
2. Найти количество мест во всех самолетах;
3. Найти количество мест во всех самолетах с учетом типа места;
4. Найти количество билетов пассажира ALEKSANDR STEPANOV, телефон которого заканчивается на 11;
5. Вывести всех пассажиров с именем ALEKSANDR, у которых количество билетов больше 2000. Отсортировать по убыванию количества билетов;
6. Вывести дни в сентябре 2017 с количество рейсов больше 500.
7. Вывести список городов, в которых несколько аэропортов
8. Сайт http://www.sql-ex.ru/learn_exercises.php. Задания: 11,12,15

Задание 3 (ответы см. в заметках к слайду)

1. Что такое dual, between, in, like. Чем отличается like от «=»
2. select sysdate from dual
3. select trunc(sysdate) from dual
4. select trunc(sysdate,'yyyy') from dual
5. select round(51236.99) from dual
6. select trunc(51236.99) from dual
7. Select substr('asdfgh', 5,1) from dual
8. Select substr('asdfgh', 4) from dual
9. Select lpad('123', 5,'0') from dual
10. Вывести модель самолета и список мест в нем
11. Вывести информацию по всем рейсам из аэропортов в г.Москва за сентябрь 2017
12. Вывести кол-во рейсов по каждому аэропорту в г.Москва за 2017
13. Вывести кол-во рейсов по каждому аэропорту, месяцу в г.Москва за 2017
14. Найти все билеты по бронированию на «3А4В»
15. Найти все перелеты по бронированию на «3А4В»
16. Читать презентацию
17. Сайт http://www.sql-ex.ru/learn_exercises.php. Задания: 6,9,13,14,16,19,20,21

Задание 4. Вложенные и коррелируемые подзапросы, exists, any, all (ответы см. в заметках к слайду)

1. Вывести все самолеты с дальность полета, такой же как дальность самолетов из архива (aircrafts_data_archive)
2. Вывести все самолеты с дальностью полета больше, чем у всех самолетов из архива
3. Вывести все самолеты с дальностью полета больше, чем у любого самолета из архива
4. Вывести все самолеты с дальностью полета, как у самолета из архива с минимальной дальность
5. Вывести все самолеты, если модели этих самолетов есть в архиве
6. Вывести информацию по самолетам. Для моделей, которые есть в архиве вывести конструктора
7. Как правило в городе есть только один аэропорт, исключение:
8. Показать данные по самолетам, у которых нет рейсов 01.09.2017
9. Показать данные по самолетам, у которых есть рейсы 01.09.2017 и в которых больше 150 мест
10. Показать все рейсы 01.09.2017 по самолетам с дальностью полета свыше 8тыс.км
11. Получить список перелетов, которые дороже всех перелетов по билету 0005432661915
12. Найти перелеты стоимость которых равна стоимости любого перелета по билету 0005432661915
13. Сайт http://www.sql-ex.ru/learn_exercises.php. Задания: 10, 17,18,21,22,23
14. Читать презентацию (стр 41-44)

Задание 5 (case, decode, union, union all, intersect, minus, cte)

1. Получить информацию по аэропортам. Если аэропорт находится в MOSCOW, то вывести город и название. Если аэропорт находится в NALCHIK, то вывести город, иначе название. (использовать decode и case);
2. Вывести название аэропорта и информацию по кол-ву рейсов. Если аэропорт находится в MOSCOW или NALCHIK, то вывести кол-во рейсов за 01.09.2017, иначе общее кол-во рейсов по аэропорту.
3. Выбрать все модели самолетов из таблиц AIRCRAFTS1 и AIRCRAFTS2
4. Выбрать все модели самолетов из таблиц AIRCRAFTS1 и AIRCRAFTS2 без повторов
5. Выбрать одинаковые модели самолетов из таблиц AIRCRAFTS1 и AIRCRAFTS2
6. Выбрать модели самолетов, которые есть в AIRCRAFTS2, но нет в AIRCRAFTS1
7. Получить список всех моделей самолетов из двух. Внизу списка вывести итоговую строку «Total N». Отсортировать по модели, итоговая строка д.б. последней.
8. См.6 переписать запрос используя CTE
9. Презентация (слайды 47,48,50,58,59)
10. Сайт http://www.sql-ex.ru/learn_exercises.php. Задания 1-30

Задание 6 (create table, drop table, insert, update, delete, grant, revoke)

1. Создать таблицу с именем table* с полями id_table* – число, code – строка, name –строка, range - число. Первичный ключ id_table*;
2. Вставить в таблицу одну строку с произвольными значениями (для поля id_table* использовать последовательность sq_id_table1.nextval);
3. Вставить данные из таблицы «Самолеты», по самолетам с кодами 619 или на 3%;
4. Добавить записи из таблицы «Самолеты», моделей который нет в архиве (aircrafts_data_archive)
5. В своей таблице обновить поле наименование по коду 320, 319, у остальных самолетов увеличить дальность полета в 2 раза;
6. Удалить данные по самолетам с кодами CN%
7. Из таблицы AIRCRAFTS_DATA, удалить данные по самолету с кодом 319.
8. Дать права на чтение из таблицы AIRCRAFTS_DATA, для пользователя user1. Подключиться под пользователем user1(пароль pass1) и проверить есть ли права.
9. Удалить права на таблицу AIRCRAFTS_DATA у пользователя user1
10. Удалить созданную таблицу table*;
11. Презентация (69-78)
12. Сайт http://www.sql-ex.ru/learn_exercises.php. Задания: 1-30

Древовидный запрос(предложение

- SQL тренажер https://www.sql-ex.ru/
- SQL Fiddle http://sqlfiddle.com/
- https://learndb.ru/courses
- Том Кайт - Oracle для профессионалов