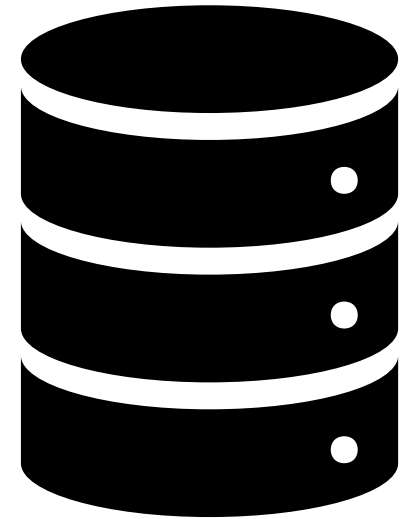


# Базы данных

Лекция 5.

Проектирование баз данных (Часть 2)

Продвинутый SQL (Часть 1)



Меркурьева Надежда

✉ [merkurievanad@gmail.com](mailto:merkurievanad@gmail.com)

📌 @merkurievanad

# Основные шаги проектирования

1. Определение данных, которые будут храниться в БД:  
Выделение предметной области и основных сущностей
2. Определение взаимосвязей между элементами данных
3. Наложение логической структуры на данные
4. Создание спроектированной базы в СУБД

# Основные этапы проектирования

- Концептуальное (инфологическое) проектирование:
  - Итог: концептуальная модель в ER-нотации
- Логическое (дatalogическое) проектирование:
  - Итог: логическая модель в ER-нотации
- Физическое проектирование
  - Итог: таблицы и взаимосвязи, созданные в базе с использованием СУБД

# Концептуальное проектирование

- Устанавливаем взаимосвязи между сущностями
- Определяем тип связи:
  - Один к одному
  - Один ко многим
  - Многие ко многим
- Строим концептуальную модель в ER-нотации со связями в нотации «воронья лапка»

# Логическое проектирование

- За основу берем имеющуюся концептуальную модель
- Детализируем сущности: выделяем атрибуты
- Дорабатываем логическую модель:
  - Пообъектно выделяем ключи и нормализуем
  - Избегаем связей с типом «один к одному» и «многие ко многим»
  - Дорабатываем типы связей после расширения списка сущностей
- Финализируем иллюстрации логической модели в ER-нотации

# Первичный ключ

- **Потенциальный ключ** – подмножество атрибутов отношения, удовлетворяющее требованиям уникальности и минимальности:
  - *Уникальность*: нет и не может быть двух кортежей данного отношения, в которых значения этого подмножества атрибутов совпадают
  - *Минимальность*: в составе потенциального ключа отсутствует меньшее подмножество атрибутов, удовлетворяющее условию уникальности
- **Первичный ключ** – это один из потенциальных ключей отношения, выбранный в качестве основного (Primary key, PK)

# Внешний ключ

Пусть  $R_1$  и  $R_2$  – две переменные отношения, не обязательно различные. **Внешним ключом FK** в  $R_2$  является подмножество атрибутов переменной  $R_2$  такое, что выполняются следующие требования:

- В переменной отношения  $R_1$  имеется потенциальный ключ РК такой, что РК и FK совпадают с точностью до переименования атрибутов
- В любой момент времени каждое значение FK в текущем значении  $R_2$  идентично значению РК в некотором кортеже в текущем значении  $R_1$ . Иными словами, в любой момент времени множество всех значений FK в  $R_2$  является подмножеством значений РК в  $R_1$ .

# Нормальные формы

- 1НФ: 1 ячейка – 1 значение
- 2НФ: 1НФ + **все** неключевые атрибуты зависят от **всех** ключевых
  - Не существует неключевого атрибута, который зависел бы от какого-либо подмножества ключевых
- 3НФ: 2НФ + **все** неключевые атрибуты зависят **только** от ключевых атрибутов:
  - Не существует неключевого атрибута, который бы зависел от какого-либо подмножества неключевых



## Шаг 4. Создание БД

- Определить, какого типа данные будут храниться
- Определить, какие ограничения накладываются на эти данные
- Создать все необходимые таблицы с использованием СУБД

# Физическая модель

*– описание реализации объектов логической модели на уровне конкретной базы данных с учетом всех ее особенностей*

# Ограничения на атрибуты

---

## NOT NULL

- Значение всегда известно, недопустимо значение NULL

## UNIQUE

- Значения в столбце должны быть уникальны

## PRIMARY KEY

- Первичный ключ. В некоторых СУБД требуется дополнительно ограничивать NOT NULL

## FOREIGN KEY

- Внешний ключ, необходима ссылка на другую таблицу

## CHECK

- Проверка на соответствие определенному критерию

## DEFAULT

- Значение по умолчанию. Используется, если пользователь не задал значения

# NOT NULL

```
CREATE TABLE PERSON (  
    ID                INTEGER          NOT NULL,  
    LAST_NAME        VARCHAR(255) NOT NULL,  
    FIRST_NAME       VARCHAR(255) NOT NULL,  
    AGE              INTEGER  
);
```

# UNIQUE

```
CREATE TABLE PERSON (  
    ID                INTEGER          NOT NULL UNIQUE,  
    LAST_NAME        VARCHAR(255)      NOT NULL,  
    FIRST_NAME       VARCHAR(255)      NOT NULL,  
    AGE              INTEGER  
);
```

```
ALTER TABLE PERSON ADD UNIQUE (ID);
```

```
ALTER TABLE PERSON  
ADD CONSTRAINT UC_Person UNIQUE (ID, LAST_NAME);
```

```
ALTER TABLE PERSON  
DROP CONSTRAINT UC_Person;
```

# PRIMARY KEY

```
CREATE TABLE PERSON (  
    ID                INTEGER          PRIMARY KEY,  
    LAST_NAME        VARCHAR(255) NOT NULL,  
    FIRST_NAME       VARCHAR(255) NOT NULL,  
    AGE              INTEGER  
);
```

```
ALTER TABLE PERSON ADD PRIMARY KEY (ID);
```

# PRIMARY KEY

```
CREATE TABLE PERSON (  
    ID                INTEGER,  
    LAST_NAME        VARCHAR(255),  
    FIRST_NAME       VARCHAR(255) NOT NULL,  
    AGE              INTEGER  
    CONSTRAINT PK_Person PRIMARY KEY (ID, LAST_NAME)  
);
```

```
ALTER TABLE PERSON  
ADD CONSTRAINT PK_Person PRIMARY KEY (ID, LastName);
```

```
ALTER TABLE PERSON  
DROP CONSTRAINT PK_Person;
```

# FOREIGN KEY

```
CREATE TABLE ORDER (  
    ORDER_ID          INTEGER NOT NULL,  
    ORDER_NUMBER      INTEGER NOT NULL,  
    PERSON_ID         INTEGER,  
    PRIMARY KEY (ORDER_ID),  
    CONSTRAINT FK_PersonOrder (PERSON_ID)  
    REFERENCES PERSON (PERSON_ID)  
);
```

```
ALTER TABLE ORDER ADD CONSTRAINT FK_PersonOrder  
FOREIGN KEY (PERSON_ID) REFERENCES PERSON (PERSON_ID);
```

```
ALTER TABLE ORDER DROP CONSTRAINT FK_PersonOrder;
```



# FOREIGN KEY

```
CREATE TABLE ORDER (  
    ORDER_ID          INTEGER NOT NULL PRIMARY KEY,  
    ORDER_NUMBER      INTEGER NOT NULL,  
    PERSON_ID         INTEGER FOREIGN KEY REFERENCES  
                        PERSON (PERSON_ID)  
);
```

```
ALTER TABLE ORDER  
ADD FOREIGN KEY (PERSON_ID)  
REFERENCES PERSON (PERSON_ID);
```

# CHECK

```
CREATE TABLE PERSON (  
    ID                INTEGER          NOT NULL,  
    LAST_NAME        VARCHAR(255) NOT NULL,  
    FIRST_NAME       VARCHAR(255) NOT NULL,  
    AGE              INTEGER CHECK (AGE >= 18)  
);
```

```
ALTER TABLE PERSON ADD CHECK (AGE>=18);
```

# CHECK

```
CREATE TABLE PERSON (  
    ID                INTEGER          NOT NULL,  
    LAST_NAME        VARCHAR(255) NOT NULL,  
    FIRST_NAME       VARCHAR(255) NOT NULL,  
    AGE              INTEGER,  
    CITY             VARCHAR(255),  
    CONSTRAINT CHK_Person CHECK (AGE >= 18  
                                   AND CITY = 'Moscow')  
);
```

```
ALTER TABLE PERSON ADD CONSTRAINT CHK_Person  
CHECK (AGE >= 18 AND CITY = 'Moscow')
```

```
ALTER TABLE PERSON DROP CONSTRAINT CHK_PersonAge;
```

# DEFAULT

```
CREATE TABLE ORDER (  
    ORDER_ID          INTEGER PRIMARY KEY,  
    ORDER_NUMBER      INTEGER NOT NULL,  
    ORDER_DATE        DATE DEFAULT now()::date  
);
```

```
ALTER TABLE ORDER  
ALTER COLUMN ORDER_DATE DROP DEFAULT;
```

При добавлении ограничений такого типа необходимо иметь в виду, что синтаксис в разных диалектах очень разный, лучше всего заглянуть в случае необходимости

# Как посмотреть ограничения в базе?

Список колонок, попадающих под ограничение:

```
SELECT *  
FROM information_schema.constraint_column_usage;
```

| table_catalog | table_schema | table_name          | column_name          | constraint_catalog | constraint_schema | constraint_name                                |
|---------------|--------------|---------------------|----------------------|--------------------|-------------------|--|
| postgres      | school       | audience_x_addition | frequency_x_addition | postgres           | school            | audience_x_addition_frequency_x_addition_check |
| postgres      | school       | audience_x_addition | audience_id          | postgres           | school            | audience_x_addition_pkey                       |
| postgres      | school       | audience_x_addition | addition_id          | postgres           | school            | audience_x_addition_pkey                       |
| postgres      | school       | audience_x_lesson   | lesson_id            | postgres           | school            | audience_x_lesson_pkey                         |
| postgres      | school       | audience_x_lesson   | audience_id          | postgres           | school            | audience_x_lesson_pkey                         |
| postgres      | studio       | ballroom            | ballroom_size        | postgres           | studio            | ballroom_ballroom_size_check                   |
| postgres      | studio       | ballroom            | ballroom_id          | postgres           | studio            | ballroom_pkey                                  |
| postgres      | studio       | ballroom            | ballroom_id          | postgres           | studio            | class_ballroom_id_fkey                         |
| postgres      | my_project   | band_x_musician     | band_id              | postgres           | my_project        | band_x_musician_pkey                           |
| postgres      | my_project   | band_x_musician     | musician_id          | postgres           | my_project        | band_x_musician_pkey                           |
| postgres      | my_project   | band_x_release      | band_id              | postgres           | my_project        | band_x_release_pkey                            |
| postgres      | my_project   | band_x_release      | release_id           | postgres           | my_project        | band_x_release_pkey                            |
| postgres      | project      | blog                | blog_id              | postgres           | project           | blog_pkey                                      |
| postgres      | project      | blog                | blog_id              | postgres           | project           | blog_comment_blog_id_fkey                      |
| postgres      | project      | blog                | blog_id              | postgres           | project           | post_blog_id_fkey                              |
| postgres      | project      | blog_comment        | comment_id           | postgres           | project           | blog_comment_pkey                              |

# Как посмотреть ограничения в базе?

Все имеющиеся в базе ограничения:

```
SELECT *  
FROM information_schema.table_constraints;
```

| constraint_catalog | constraint_schema | constraint_name                 | 1 | table_catalog | table_schema | table_name       | constraint_type | is_deferrable | initially_deferred |
|--------------------|-------------------|---------------------------------|---|---------------|--------------|------------------|-----------------|---------------|--------------------|
| postgres           | pg_catalog        | 11_3381_5_not_null              |   | postgres      | pg_catalog   | pg_statistic_ext | CHECK           | NO            | NO                 |
| postgres           | pg_catalog        | 11_3381_6_not_null              |   | postgres      | pg_catalog   | pg_statistic_ext | CHECK           | NO            | NO                 |
| postgres           | motoservice       | _client_client_rating_check     |   | postgres      | motoservice  | _client_         | CHECK           | NO            | NO                 |
| postgres           | motoservice       | _client_client_reports_check    |   | postgres      | motoservice  | _client_         | CHECK           | NO            | NO                 |
| postgres           | motoservice       | _client_pkey                    |   | postgres      | motoservice  | _client_         | PRIMARY KEY     | NO            | NO                 |
| postgres           | motoservice       | _crash_crash_complexity_check   |   | postgres      | motoservice  | _crash_          | CHECK           | NO            | NO                 |
| postgres           | motoservice       | _crash_crash_fixingprice_check  |   | postgres      | motoservice  | _crash_          | CHECK           | NO            | NO                 |
| postgres           | motoservice       | _crash_pkey                     |   | postgres      | motoservice  | _crash_          | PRIMARY KEY     | NO            | NO                 |
| postgres           | motoservice       | _department_pkey                |   | postgres      | motoservice  | _department_     | PRIMARY KEY     | NO            | NO                 |
| postgres           | motoservice       | _makers_pkey                    |   | postgres      | motoservice  | _makers_         | PRIMARY KEY     | NO            | NO                 |
| postgres           | motoservice       | _vehicle_maker_id_fkey          |   | postgres      | motoservice  | _vehicle_        | FOREIGN KEY     | NO            | NO                 |
| postgres           | motoservice       | _vehicle_pkey                   |   | postgres      | motoservice  | _vehicle_        | PRIMARY KEY     | NO            | NO                 |
| postgres           | motoservice       | _vehicle_worker_id_fkey         |   | postgres      | motoservice  | _vehicle_        | FOREIGN KEY     | NO            | NO                 |
| postgres           | motoservice       | _vehicle_client_clients_id_fkey |   | postgres      | motoservice  | _vehicle_client_ | FOREIGN KEY     | NO            | NO                 |
| postgres           | motoservice       | _vehicle_client_pkey            |   | postgres      | motoservice  | _vehicle_client_ | PRIMARY KEY     | NO            | NO                 |
| postgres           | motoservice       | _vehicle_client_vehicle_id_fkey |   | postgres      | motoservice  | _vehicle_client_ | FOREIGN KEY     | NO            | NO                 |
| postgres           | motoservice       | _vehicle_crash_crash_id_fkey    |   | postgres      | motoservice  | _vehicle_crash_  | FOREIGN KEY     | NO            | NO                 |
| postgres           | motoservice       | _vehicle_crash_pkey             |   | postgres      | motoservice  | _vehicle_crash_  | PRIMARY KEY     | NO            | NO                 |

# Как посмотреть ограничения в базе?

Уникальные и ключевые (PK, FK) поля таблиц

```
SELECT *  
FROM information_schema.key_column_usage;
```

| constraint_catalog | constraint_schema | constraint_name                     | table_catalog | table_schema | table_name         | column_name     | ordinal_position |
|--------------------|-------------------|-------------------------------------|---------------|--------------|--------------------|-----------------|------------------|
| postgres           | studio            | client_pkey                         | postgres      | studio       | client             | client_id       | 1                |
| postgres           | studio            | pk_class_x_client                   | postgres      | studio       | class_x_client     | client_id       | 1                |
| postgres           | studio            | class_x_client_client_id_fkey       | postgres      | studio       | class_x_client     | client_id       | 1                |
| postgres           | motoservice       | _client_pkey                        | postgres      | motoservice  | _client            | client_id       | 1                |
| postgres           | clinic            | client_pkey                         | postgres      | clinic       | client             | client_id       | 1                |
| postgres           | clinic            | appointment_client_id_fkey          | postgres      | clinic       | appointment        | client_id       | 1                |
| postgres           | motoservice       | _vehicle_client_pkey                | postgres      | motoservice  | _vehicle_client    | clients_id      | 2                |
| postgres           | motoservice       | _vehicle_client_clients_id_fkey     | postgres      | motoservice  | _vehicle_client    | clients_id      | 1                |
| postgres           | project           | blog_comment_pkey                   | postgres      | project      | blog_comment       | comment_id      | 1                |
| postgres           | public            | competition_pkey                    | postgres      | public       | competition        | competition_id  | 1                |
| postgres           | public            | pk_result                           | postgres      | public       | result             | competition_id  | 1                |
| postgres           | public            | result_competition_id_fkey          | postgres      | public       | result             | competition_id  | 1                |
| postgres           | public            | consultation_pkey                   | postgres      | public       | consultation       | consultation_id | 1                |
| postgres           | project           | contractor_pkey                     | postgres      | project      | contractor         | contractor_id   | 1                |
| postgres           | project           | pk_store_contractor                 | postgres      | project      | store_x_contractor | contractor_id   | 1                |
| postgres           | project           | store_x_contractor_contractor_id... | postgres      | project      | store_x_contractor | contractor_id   | 1                |

# Как посмотреть ограничения в базе?

Информация по ограничениям с типом CHECK

```
SELECT *  
FROM information_schema.check_constraints;
```

| constraint_catalog | constraint_schema  | constraint_name                         | check_clause   |
|--------------------|--------------------|---|--|
| postgres           | clinic             | consulting_room_floor_no_check          | ((floor_no > 0))   |
| postgres           | clinic             | service_price_check                     | ((price > 0))  |
| postgres           | clinic             | 35963_36031_2_not_null                  | doctor_id IS NOT NULL  |
| postgres           | clinic             | 35963_35991_1_not_null                  | client_id IS NOT NULL  |
| postgres           | clinic             | 35963_35996_1_not_null                  | appointment_id IS NOT NULL                                     |
| postgres           | clinic             | 35963_36016_2_not_null                  | room_no IS NOT NULL  |
| postgres           | clinic             | 35963_35964_1_not_null                  | room_no IS NOT NULL  |
| postgres           | information_schema | yes_or_no_check                         | ((VALUE)::text = ANY ((ARRAY['YES'::character varying, 'NO'... |
| postgres           | information_schema | cardinal_number_domain_check            | ((VALUE >= 0))   |
| postgres           | institute          | research_research_budget_check          | ((research_budget >= 0))                                       |
| postgres           | institute          | 47412_47422_1_not_null                  | lab_id IS NOT NULL   |
| postgres           | institute          | equipment_x_lab_equipment_lab_cnt_check | ((equipment_lab_cnt >= 0))                                     |
| postgres           | institute          | 47412_47413_2_not_null                  | scientist_lab_id IS NOT NULL                                   |



# Уже умеем:

- Писать обычные запросы
- Писать запросы с использованием нескольких таблиц:
  - JOIN
  - Перечисление через запятую
- Оперировать с однотипными результатами запросов:
  - UNION (ALL)
  - INTERSECT
  - EXCEPT

# Хотим научиться

- Писать запросы со сложными условиями
- Писать запросы с подзапросами
- Писать аналитические запросы

# Запросы с условиями

- В SQL имеется возможность использования условных выражений
- Эта возможность реализуется с использованием ***CASE-выражений***
- CASE-выражения:
  - Простые
  - С поиском

# Запросы с условиями

## Простое CASE-выражение

```
CASE expression  
  WHEN condition_1 THEN result_1  
  WHEN condition_2 THEN result_2  
  ...  
  WHEN condition_N THEN result_N  
  ELSE result  
END
```

# Запросы с условиями

```
CASE ProductLine  
  WHEN 'R' THEN 'Road'  
  WHEN 'M' THEN 'Mountain'  
  WHEN 'T' THEN 'Touring'  
  ELSE 'Not for sale'  
END
```

# Запросы с условиями

CASE-выражение с поиском

**CASE**

**WHEN** *boolean\_expr\_1* **THEN** *result\_1*

**WHEN** *boolean\_expr\_2* **THEN** *result\_2*

...

**WHEN** *boolean\_expr\_N* **THEN** *result\_N*

**ELSE** *result*

**END**

# Запросы с условиями

## **CASE**

**WHEN** ListPrice = 0 **THEN** 'Not for resale'

**WHEN** ListPrice < 50 **THEN** 'Under \$50'

**WHEN** ListPrice >= 50

**AND** ListPrice < 250 **THEN** 'Under \$250'

**WHEN** ListPrice >= 250

**AND** ListPrice < 1000 **THEN** 'Under \$1000'

**ELSE** 'Over \$1000'

**END**

# Запросы с условиями

```
SELECT OrderID,  
        Quantity,  
        CASE  
            WHEN Quantity > 30  
            THEN "The quantity is greater than 30"  
            WHEN Quantity = 30  
            THEN "The quantity is 30"  
            ELSE "The quantity is something else"  
        END AS Comment  
FROM OrderDetails;
```



# Запросы с условиями

```
SELECT CustomerName,  
        City,  
        Country  
FROM Customers  
ORDER BY CASE  
        WHEN City IS NULL  
        THEN Country  
        ELSE City  
END;
```

# GREATEST / LEAST

- Функции *greatest()* и *least()* возвращают наибольшее и наименьшее значения соответственно
- В зависимости от диалекта работа с NULL значениями может отличаться. PostgreSQL вернет NULL только если все аргументы NULL

```
SELECT value_1, value_2
        greatest(value_1, value_2) AS gr,
        least(value_1, value_2)    AS lst
FROM T;
```

# GREATEST / LEAST

| VALUE_1 | VALUE_2 | GR   | LST  |
|---------|---------|------|------|
| 10      | 15      | 15   | 10   |
| 10      | NULL    | 10   | 10   |
| 28      | 13      | 28   | 13   |
| NULL    | 16      | 16   | 16   |
| NULL    | NULL    | NULL | NULL |

# Подзапрос

- **Подзапрос** – это запрос, содержащийся в другом SQL-выражении (будем называть его **содержащим выражением**)
- Подзапрос всегда заключен в круглые скобки и обычно выполняется до содержащего выражения
- Подзапросы могут вкладываться друг в друга
- В операторе SELECT подзапросы можно использовать во всех разделах, кроме GROUP BY

# Подзапросы

- Подзапросы бывают:
  - **Несвязанными**, т.е. полностью самостоятельными и не зависящими от основного запроса
    - Выполняются перед выполнением содержащего выражения.
  - **Связанными**, т.е. ссылаются на столбцы основного запроса.
    - Для написания таких запросов полезно использование алиасов.
    - Для случаев, когда в основном запросе и в подзапросе используется одна и та же таблица, использование алиасов обязательно!
    - Выполняются для каждой строки содержащего выражения.

# Результат подзапроса

- Результатом выполнения подзапроса всегда является таблица, которая может состоять из:
  - 1 столбца и 1 строки (*скалярный подзапрос*)
  - 1 столбца и нескольких строк
  - Нескольких столбцов

# Использование подзапросов

| Раздел   | Связанные | Несвязанные | Нескалярные |
|----------|-----------|-------------|-------------|
| SELECT   | +         | +           | -           |
| FROM     | -         | +           | +           |
| WHERE    | +         | +           | +           |
| HAVING   | +         | +           | +           |
| ORDER BY | +         | +           | -           |

# Скалярный подзапрос

Подзапрос называется **скалярным**, если результат его выполнения – таблица из 1 столбца и 1 строки.

```
SELECT  account_id,  
         product_cd,  
         cust_id,  
         avail_balance  
FROM    account  
WHERE   open_emp_id <>  
        (SELECT e.emp_id  
         FROM   employee e  
         WHERE  e.title = 'Head Teller');
```



# Скалярный подзапрос

- Подзапрос записывается следующим образом:  
    <скалярная форма> <оператор> <подзапрос>
- Т.е. следующая запись неверна, хотя некоторые реализации и позволяют ее использование:

```
SELECT account_id,  
        product_cd,  
        cust_id,  
        avail_balance  
FROM account  
WHERE (SELECT e.emp_id  
        FROM employee e  
        WHERE e.title = 'Head Teller') <> open_emp_id;
```

# Скалярный подзапрос

- Подзапрос называется **скалярным**, если результат его выполнения – таблица из 1 столбца и 1 строки.
- Если в результате выполнения подзапроса не было отобрано ни одного значения, основной запрос будет рассматривать итог как неизвестный (NULL)

# Подзапрос: 1 столбец и несколько строк

- Обычно такие подзапросы используются со следующими предикатами:
  - EXISTS
  - IN
  - ALL
  - ANY (SOME)

# Предикат EXISTS

Значением условия EXISTS является TRUE в том и только в том случае, когда мощность таблицы-результата подзапроса больше нуля, иначе значением условия является FALSE

```
SELECT SupplierName
FROM Suppliers
WHERE EXISTS
    (SELECT ProductName
     FROM Products
     WHERE SupplierId = Suppliers.supplierId
     AND Price < 20);
```

# Предикат IN

Предикат IN для подзапросов работает так же, как и для обычных запросов

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

# Предикат ALL

- Имеет значение TRUE в том и только в том случае, когда результат подзапроса пуст или значение предиката равное TRUE для каждой строки подзапроса
- Имеет значение FALSE в том и только в том случае, когда значение предиката равно FALSE хотя бы для одной из строк подзапроса
- В остальных случаях значение условия равно UNKNOWN

```
SELECT EMP_NO
FROM EMP
WHERE DEPT_NO = 65
AND EMP_SAL >= ALL
    (SELECT EMP1.EMP_SAL
     FROM EMP EMP1
     WHERE EMP.DEPT_NO = EMP1.DEPT_NO) ;
```

# Предикат ANY (SOME)

- Имеет значение FALSE в том и только в том случае, когда результат подзапроса пуст или значение условия равно FALSE для каждой строки подзапроса
- Имеет значение TRUE в том и только в том случае, когда значение предиката равно TRUE хотя бы для одной из строк подзапроса
- В остальных случаях значение условия равно UNKNOWN

```
SELECT EMP_NO
FROM EMP
WHERE DEPT_NO = 65
AND EMP_SAL > ANY
    (SELECT EMP1.EMP_SAL
     FROM EMP EMP1
     WHERE EMP.DEPT_NO = EMP1.DEPT_NO);
```

# Несколько столбцов

- При использовании такого запроса второй операнд предиката должен представлять собой список выражений, приведенный через запятую в скобках
- Тип допустимых предикатов определяется количеством строк, которые возвращает подзапрос
- Если при использовании предикатов сравнения подзапрос возвращает более одной строки, возникает ошибка
- Подзапрос действует как временная таблица, областью видимости которой является выражение



# Несколько столбцов

```
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, employee e  
        WHERE b.branch_id = e.assigned_branch_id  
          AND b.name = 'Woburn Branch'  
          AND e.title = 'Head Teller');
```

# CREATE AS

```
CREATE TABLE NEW_TABLE AS  
SELECT *  
    FROM OLD_TABLE;
```

```
CREATE TABLE NEW_TABLE AS  
SELECT ATTR_1,  
        ATTR_2  
    FROM OLD_TABLE  
WHERE ATTR_1 > 100  
    AND ATTR_2 < 50;
```

# CREATE AS

```
CREATE TABLE NEW_TABLE AS  
SELECT ATTR_1, ATTR_2,..., ATTR_N  
      FROM TABLE_1, TABLE_2,..., TABLE_N;
```

# CREATE AS

```
CREATE TABLE NEW_TABLE AS  
SELECT *  
    FROM OLD_TABLE  
WHERE 1 = 2;
```

# CREATE AS

```
CREATE TABLE NEW_TABLE AS  
SELECT *  
    FROM OLD_TABLE  
WHERE 1 = 2;
```

Получим такую же структуру, как у старой таблицы, но не скопируем ни одного значения

# Аналитические (оконные) функции

- Аналитические функции принимают в качестве аргумента столбец промежуточного результата вычисления и возвращают тоже столбец
- Местом их использования могут быть только разделы ORDER BY и SELECT, выполняющие завершающую обработку логического промежуточного результата
- Аналитические функции действуют подобно агрегатным функциям, но не уменьшают степень детализации
- Аналитические функции агрегируют данные порциями, количество и размер которых регулируется специальной синтаксической конструкцией

# Аналитические функции

```
an_function(expression) OVER (  
    [PARTITION BY expression_comma_list_1]  
    [ORDER BY expression_comma_list_2 [{ASC | DESC}]]  
    [{ROWS | RANGE} {(UNBOUNDED | expression_0) PRECEDING |  
    CURRENT ROW}  
    |  
    {ROWS | RANGE}  
    BETWEEN  
    {{UNBOUNDED PRECEDING | CURRENT ROW |  
    {UNBOUNDED | expression_1} {PRECEDING | FOLLOWING}}  
    AND  
    {{UNBOUNDED FOLLOWING | CURRENT ROW |  
    {UNBOUNDED | expression_2} {PRECEDING | FOLLOWING}}  
    ] )
```

# Использование OVER

- OVER определяет «окно» или набор строк, которые будет использовать оконная функция, включая сортировку данных
- В выражении, которое задает оконную функцию, инструкция OVER ограничивает наборы строк с одинаковыми значениями в поле, по которому идет разделение
- Сама по себе инструкция OVER ( ) не ограничена и содержит все строки из результирующего набора
- Инструкция OVER может многократно использоваться в одном SELECT, каждая со своим разделением и сортировкой



# Использование OVER

```
OVER ( [ <PARTITION BY clause> ]  
        [ <ORDER BY clause> ]  
        [ <ROWS or RANGE clause> ]  
      )
```

# Правила секционирования

- Внутри OVER необходимо указать поле таблицы по которому будет скользить «окно» и правило по которому строки будут секционироваться:
  - PARTITION BY
  - ORDER BY
  - ROWS | RANGE

# PARTITION BY

- Логически разбивает множество на группы по критериям
- Аналитические функции применяются к группам независимо
- Если не указать конструкцию секционирования, все множество считается одной группой

# ORDER BY

- Задаёт критерий сортировки внутри каждой группы
- Агрегатные функции в отсутствие конструкции ORDER BY вычисляются по всем строкам группы, и одно и то же значение выдается для каждой строки, т.е. функция используется как итоговая
- Если агрегатная функция используется с конструкцией ORDER BY, то она вычисляется по текущей строке и всем строкам до неё, т.е. функция используется как оконная

# PARTITION BY

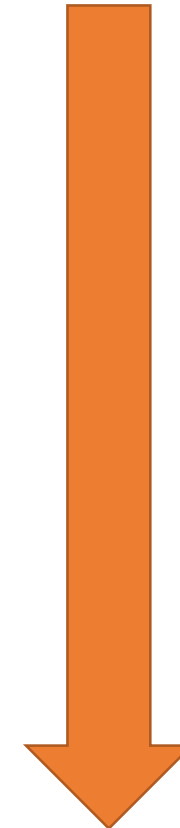
```
SELECT TerritoryID, SalesYTD,  
       SUM(SalesYTD) OVER (PARTITION BY TerritoryID) AS CumulativeTotal,  
       SalesYTD / SUM(SalesYTD) OVER (PARTITION BY TerritoryID) * 100 AS SalesTotalPrc  
FROM SalesPerson;
```

| TerritoryId | SalesYTD     | CumulativeTotal | SalesTotalPrc |        |
|-------------|--------------|-----------------|---------------|--------|
| NULL        | 559697,5639  | 1252127,9471    | 44,69         | Окно 1 |
| NULL        | 172524,4512  | 1252127,9471    | 13,77         |        |
| NULL        | 519905,932   | 1252127,9471    | 41,52         |        |
| 1           | 1573012,9383 | 4502152,2674    | 34,39         | Окно 2 |
| 1           | 1576562,1966 | 4502152,2674    | 35,01         |        |
| 1           | 1352577,1325 | 4502152,2674    | 30,04         |        |
| 2           | 3763178,1787 | 3763178,1787    | 100,0         | Окно 3 |
| 3           | 3189418,3662 | 3189418,3662    | 100,0         | Окно 4 |
| 4           | 4251368,5497 | 6709904,1666    | 63,35         | Окно 5 |
| 4           | 2458535,6169 | 6709904,1666    | 36,65         |        |

# ORDER BY

```
SELECT TerritoryID, SalesYTD,  
       SUM(SalesYTD) OVER (ORDER BY TerritoryID ASC) AS CumulativeTotal  
FROM SalesPerson;
```

| TerritoryId | SalesYTD     | CumulativeTotal |
|-------------|--------------|-----------------|
| NULL        | 559697,5639  | 1252127,9471    |
| NULL        | 172524,4512  | 1252127,9471    |
| NULL        | 519905,932   | 1252127,9471    |
| 1           | 1573012,9383 | 5754280,2145    |
| 1           | 1576562,1966 | 5754280,2145    |
| 1           | 1352577,1325 | 5754280,2145    |
| 2           | 3763178,1787 | 9517458,3932    |
| 3           | 3189418,3662 | 12706876,7594   |
| 4           | 4251368,5497 | 19416780,926    |
| 4           | 2458535,6169 | 19416780,926    |



Нарастающий  
итог