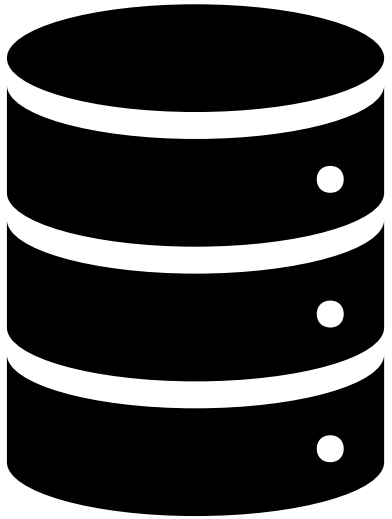


Базы данных

Лекция 6.

Продвинутый SQL: Оконные функции.

Типовые задачи SQL.

Меркурьева Надежда
✉ merkurievanad@gmail.com
@merkurievanad

Аналитические (оконные) функции

- Принимают в качестве аргумента столбец промежуточного результата вычисления
- Возвращают тоже столбец
- Местом их использования могут быть только разделы:
  - ORDER BY
  - SELECT
- Похожи на агрегатные функциям, но не уменьшают степень детализации
- 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 набор строк, которые будет использовать оконная функция («окно»)

OVER ограничивает окно наборами строк с одинаковыми значениями в полях, обозначенных в блоке **PARTITION BY**

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> ]
) AS attr_name
```

Правила секционирования

- Внутри OVER необходимо указать поле таблицы, по которому будет скользить «окно» и правило по которому строки будут секционироваться:
  - `PARTITION BY – отвечает за критерий секционирования`
  - `ORDER BY – отвечает за сортировку`
  - `ROWS | RANGE – дополнительные ограничения на диапазон строк окна (обязательно присутствие ORDER BY)`

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

- Задает критерий сортировки внутри каждой группы
- Агрегатные функции (`COUNT, MIN, MAX, SUM, AVG`) 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	44.7	Окно 1
NULL	172524.4512	1252127.9471	13.78	
NULL	519905.932	1252127.9471	41.52	
1	1573012.9383	4502152.2674	34.94	Окно 2
1	1576562.1966	4502152.2674	35.02	
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	63.36	Окно 5
4	2458535.6169	6709904.1666	36.64	

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

ROWS

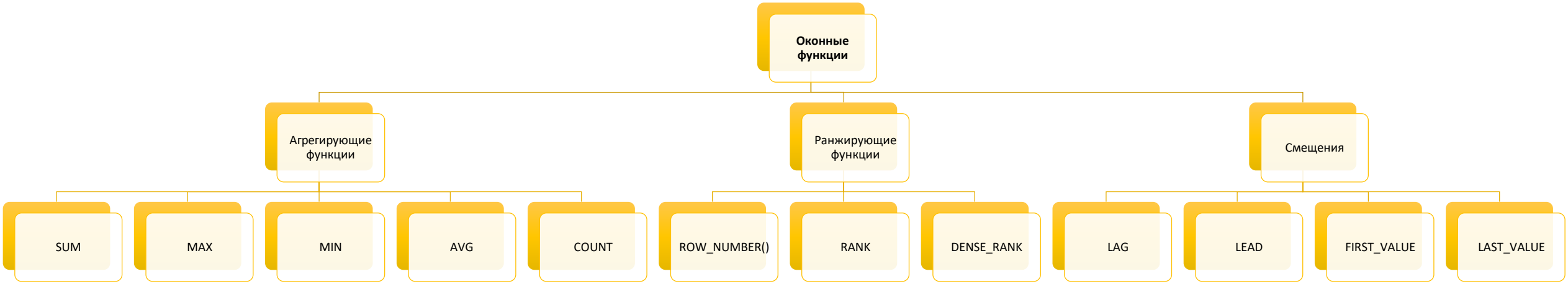
```
SELECT  TerritoryID, SalesYTD,
        SUM(SalesYTD) OVER (ORDER BY TerritoryID ASC ROWS BETWEEN CURRENT ROW AND 1
        FOLLOWING) AS CumulativeTotal,
  FROM SalesPerson;
```

TerritoryID	SalesYTD	CumulativeTotal	
NULL	559697.5639	732222.0151	Окно 1
NULL	172524.4512	692430.3832	
NULL	519905.932	2092918.8703	
1	1573012.9383	3149575.1349	Окно 2
1	1576562.1966	2929139.3291	
1	1352577.1325	5115755.3112	
2	3763178.1787	6952596.5449	
3	3189418.3662	7440786.9159	
4	4251368.5497	6709904.1666	Окно N-1
4	2458535.6169	2458535.6169	Окно N

RANGE

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

TerritoryID	SalesYTD	CumulativeTotal	
NULL	172524.4512	1252127.9471	Окно 1
NULL	519905.9320	1252127.9471	
NULL	559697.5639	1252127.9471	
1	1352577.1325	4502152.2674	Окно 2
1	1573012.9383	4502152.2674	
1	1576562.1966	4502152.2674	
2	3763178.1787	3763178.1787	Окно 3
3	3189418.3662	3189418.3662	Окно 4
4	2458535.6169	6709904.1666	Окно 5
4	4251368.5497	6709904.1666	



Оконные функции

Агрегирующие оконные функции

```
SELECT BusinessEntity, TerritoryID, SalesYear, SalesYTD,
       SUM(SalesYTD) OVER (PARTITION BY TerritoryID) AS TotalByTerritory,
       SUM(SalesYTD) OVER (PARTITION BY TerritoryID ORDER BY SalesYear) AS TotalByTerritoryYear
  FROM SalesPerson
 ORDER BY TerritoryID, SalesYear;
```

BusinessEntity	TerritoryID	SalesYear	SalesYTD	TotalByTerritory	TotalByTerritoryYear
274	NULL	2005	559 697,5639	1 252 127,9471	559 697,5639
287	NULL	2006	172 524,4512	1 252 127,9471	732 222,0151
285	NULL	2007	519 905,932	1 252 127,9471	1 252 127,9471
283	1	2005	1 573 012,9383	4 502 152,2674	3 149 575,1349
280	1	2005	1 576 562,1966	4 502 152,2674	3 149 575,1349
284	1	2006	1 352 577,1325	4 502 152,2674	4 502 152,2674
275	2	2005	3 763 178,1787	3 763 178,1787	3 763 178,1787
277	3	2005	3 189 418,3662	3 189 418,3662	3 189 418,3662
276	4	2005	4 251 368,5497	6 709 904,1666	6 709 904,1666
281	4	2005	2 458 535,6169	6 709 904,1666	6 709 904,1666

Нарастающий итог

Нарастающий итог по годам

Ранжирующие оконные функции

- *row_number() – нумеруем каждую строку окна последовательно с шагом 1*
- *rank() – ранжируем каждую строку окна с разрывом в нумерации при равенстве значений*
- *dense_rank() – ранжируем каждую строку окна без разрывов в нумерации при равенстве значений*

Ранжирующие оконные функции

```
SELECT p.FirstName, p.LastName,
       a.PostalCode,
       ROW_NUMBER() OVER (ORDER BY
       a.PostalCode) AS "Row Number",
       RANK() OVER (ORDER BY a.PostalCode)
       AS "Rank"
       DENSE_RANK() OVER (ORDER BY
       a.PostalCode) AS "Dense Rank"
  FROM Person AS p
       INNER JOIN Address AS a
               ON a.AddressID = p.AddressID;
```

FirstName	LastName	PostalCode	Row Number	Rank	Dense Rank
Michael	Blythe	98027	1	1	1
Linda	Mitchell	98027	2	1	1
Jillian	Carson	98027	3	1	1
Garrett	Vargas	98027	4	1	1
Tsvi	Reiter	98027	5	1	1
Pamela	Ansman-Wolfe	98027	6	1	1
Shu	Ito	98055	7	7	2
José	Saraiva	98055	8	7	2
David	Campbell	98055	9	7	2
Tete	Mensa-Annan	98055	10	7	2
Lynn	Tsoflias	98055	11	7	2
Rachel	Valdez	98055	12	7	2
Jae	Pak	98055	13	7	2
Ranjit	Varkey Chudukatil	98055	14	7	2

Порядковый номер строки

Разрыв послед-ти

Порядковый номер группы

Функции смещения

- *lag(attr, offset, default_value) – предыдущее значение со сдвигом*
- *lead(attr, offset, default_value) – следующее значение со сдвигом*
- *first_value(attr) – первое значение в окне с первой по текущую строку*
- *last_value(attr) – последнее значение в окне с первой по текущую строку*

Функции смещения

```
SELECT BusinessEntity, SalesYear, CurrentQuota,
       LAG(CurrentQuota, 1, 0) OVER (ORDER BY SalesYear) AS PrevQuota
       LEAD(CurrentQuota, 1, 0) OVER (ORDER BY SalesYear) AS NextQuota
  FROM SalesPersonQuotaHistory
 WHERE BusinessEntityID = 275;
```

BusinessEntity	SalesYear	CurrentQuota	PrevQuota	NextQuota
275	2005	367000	0	556000
275	2005	556000	367000	502000
275	2006	502000	556000	550000
275	2006	550000	502000	1429000
275	2006	1429000	550000	1324000
275	2006	1324000	1429000	0

Типовые задачи SQL

- Поддержка версионности таблиц
- Соединение версионных таблицы
- Поиск разрывов в версионности
- Поиск пересечений в версионности
- Сравнение таблиц

Slowly Changing Dimensions (SCD)

- ***Slowly changing dimensions (SCD)*** – редко изменяющиеся измерения, то есть измерения, не ключевые атрибуты которых имеют тенденцию со временем изменяться
- Выделяют 5 основных типов (нумерация с 0)

SCD: тип 0

- После попадания в таблицу данные никогда не изменяются

- Практически никогда не используется (по понятным причинам)
- Не поддерживает версионность
- Является начальной «точкой отсчета» методологии SCD

SCD: тип 1

- Данные записываются поверх существующих значений
- Старые значения нигде не сохраняются
- Используется, если история не нужна

- Достоинства:
  - Не добавляется избыточность
  - Очень простая структура
- Недостатки:
  - Не хранит историю

SCD: тип 2

- Создание новой записи в таблице под каждую версию данных с добавлением полей даты начала и даты конца периода существования версии

NAME	POSITION_ID	DEPT	DATE_START	DATE_END
Николай	21	2	2010-08-11	9999-12-31
Денис	23	3	2010-08-11	9999-12-31
Борис	26	2	2010-08-11	9999-12-31
Пенни	25	2	2010-08-11	9999-12-31

SCD: тип 2

- В полях `START_DT` и `END_DT` обычно не используются значения `NULL`
- Вместо NULL используется некоторая константа, например, `'9999-12-31'` для `END_DT`, как в примере
- Такой подход упрощает написание условий:

```
WHERE snapshot_date BETWEEN DATE_START AND DATE_END
```

вместо

```
WHERE snapshot_date >= DATE_START
  AND (snapshot_date <= DATE_END
       OR DATE_END IS NULL)
```

SCD: тип 2

- Достоинства:
  - Хранит полную и неограниченную историю версий
  - Удобный и простой доступ к данным необходимого периода
- Недостатки:
  - Провоцирует на избыточность или заведение дополнительных таблиц для хранения изменяемых атрибутов

SCD: тип 3

- В самой записи содержатся дополнительные поля для предыдущих значений атрибута.
- При получении новых данных, старые данные перезаписываются текущими значениями.

ID	UPDATE_TIME	LAST_STATE	CURRENT_STATE
1	11.08.2010 12:58	0	1
2	11.08.2010 12:29	1	1

SCD: тип 3

- Достоинства:
  - Небольшой объем данных
  - Простой и быстрый доступ к истории
- Недостатки:
  - Ограниченная история

SCD: тип 4

- История изменений содержится в отдельной таблице: основная таблица всегда перезаписывается текущими данными с перенесением старых данных в другую таблицу.
- Обычно этот тип используют для аудита изменений или создания архивных таблиц.

SCD: тип 4

Таблица с актуальными данными

NAME	POSITION_ID	DEPT
Коля	21	2
Денис	23	3
Борис	26	2
Пенни	25	2

Таблица с историей

NAME	POSITION_ID	DEPT	DATE
Коля	21	1	11.08.2010 14:12
Денис	23	2	11.08.2010 14:12
Борис	26	1	11.08.2010 14:12

SCD: тип 4

- Достоинства:
  - Быстрая работа с текущими версиями
- Недостатки:
  - Разделение единой сущности на разные таблицы

Версионность

- Наиболее используемый тип SCD – тип 2

- Как соединять такие таблицы с версионными данными?

Версионность

Имеем:

EMP_DEPT

EMP_NAME	DEPT_NO	START_DATE	END_DATE
SMITH	1	1995-05-01	2000-06-15
SMITH	2	2000-06-16	9999-12-31
WARD	1	2001-05-10	9999-12-31
JONES	2	1999-03-05	2001-04-25
BLAKE	1	1996-07-20	1997-01-15
BLAKE	2	1997-01-16	2002-05-17
BLAKE	3	2002-05-18	9999-12-31

DEPT

DEPT_NO	DEPT_NAME	START_DATE	END_DATE
1	Департамент 1_1	1995-01-01	1997-12-31
1	Департамент 1_2	1998-01-01	9999-12-31
2	Департамент 2_1	1995-01-01	1997-12-31
2	Департамент 2_2	1998-01-01	9999-12-31
3	Департамент 3_1	2002-01-01	9999-12-31

Хотим получить:

EMP_NAME	DEPT_NAME	START_DATE	END_DATE
SMITH	Департамент 1_1	1995-05-01	1997-12-31
SMITH	Департамент 1_2	1998-01-01	2000-06-15
SMITH	Департамент 2_2	2000-06-16	9999-12-31
WARD	Департамент 1_2	2001-05-10	9999-12-31
JONES	Департамент 2_1	1999-03-05	1997-12-31
JONES	Департамент 2_2	1998-01-01	2001-04-25
BLAKE	Департамент 1_1	1996-07-20	1997-01-15
BLAKE	Департамент 2_1	1997-01-16	1997-12-31
BLAKE	Департамент 2_2	1998-01-01	2002-05-17
BLAKE	Департамент 3_1	2002-05-18	9999-12-31

Условия соединения

- Строки из EMP_DEPT должны соединяться только со строками соответствующего отдела из DEPT:
  - `EMP_DEPT.DEPT_NO = DEPT.DEPT_NO`
- Конкретная строка из таблицы EMP_DEPT (исходная строка) должна соединяться только с теми строками таблицы DEPT (соединяемая строка), у которых период действия покрывает часть периода действия исходной строки:
  - `DEPT.START_DATE <= EMP_DEPT.END_DATE`
  - **`AND`** `DEPT.END_DATE >= EMP_DEPT.START_DATE`
- Датой начала действия результирующего интервала должна быть максимальная дата из дат начала действия исходного и присоединяемого интервала:
  - `START_DATE = max(EMP_DEPT.START_DATE, DEPT.START_DATE)`
- Датой окончания действия результирующего интервала должна быть минимальная дата из дат окончания действия исходного и присоединяемого интервала:
  - `END_DATE = `**`min`**`(EMP_DEPT.END_DATE, DEPT.END_DATE)`

Соединение версионных таблиц

```
SELECT EMP_DEPT.EMD_NAME,
       DEPT.DEPT_NAME,
       CASE
         WHEN EMP_DEPT.START_DATE > DEPT.START_DATE
         THEN EMP_DEPT.START_DATE
         ELSE DEPT.START_DATE
       END AS START_DATE,
       CASE
         WHEN EMP_DEPT.END_DATE < DEPT.END_DATE
         THEN EMP_DEPT.END_DATE
         ELSE DEPT.END_DATE
       END AS END_DATE
  FROM EMP_DATE
       INNER JOIN DEPT
               ON EMP_DATE.DEPT_NO = DEPT.DEPT_NO
              AND DEPT.START_DATE <= EMP_DEPT.END_DATE
              AND DEPT.END_DATE >= EMP_DEPT.START_DATE;
```

Поиск разрывов и пересечений в версионности

- Версионные таблицы хранят историю изменения атрибутов
- История изменения атрибутов представляет собой набор временных интервалов
- Зачастую требуется проверка истории на наличие разрывов или пересечений в версионности

Поиск разрывов в версионности

EMP_NAME	DEPT_NO	START_DATE	END_DATE
SMITH	1	1995-05-01	2000-06-15
SMITH	2	2000-06-16	9999-12-31
JONES	1	1996-05-10	1997-11-10
JONES	2	1999-03-05	2001-04-25
BLAKE	1	1996-07-20	1997-01-15
BLAKE	2	1997-01-16	2002-05-15
BLAKE	3	2002-05-18	9999-12-31

Поиск разрывов в версионности

EMP_NAME	DEPT_NO	START_DATE	END_DATE
SMITH	1	1995-05-01	2000-06-15
SMITH	2	2000-06-16	9999-12-31
JONES	1	1996-05-10	1997-11-10
JONES	2	1999-03-05	2001-04-25
BLAKE	1	1996-07-20	1997-01-15
BLAKE	2	1997-01-16	2002-05-15
BLAKE	3	2002-05-18	9999-12-31

В таблице присутствуют 2 разрыва:

1. Для сотрудника ‘JONES’ нет информации о его работе в период с ‘1997-11-11’ по ‘1999-03-04’
2. Для сотрудника ‘BLAKE’ нет информации о его работе в период с ‘2002-05-16’ по ‘2002-05-17’

Разрывы

- Логика исходных данных
  - сотрудник ‘JONES’ не работал в компании с ‘1997-11-11’ по ‘1999-03-04’
- Ошибка ввода данных
  - у сотрудника ‘BLAKE’ неправильно введена дата начала его работы в отделе 3 или дата окончания его работы в отделе 2

Поиск разрывов в версионности

- Использование аналитических функций
- Разбить данные на группы по ключу таблицы:
  - по `EMP_NAME`
- Отсортировать данные в группах по дате начала действия интервала:
  - `START_DATE`
- Проверить, превышает ли разница между датой начала действия текущего интервала и датой окончания действия предыдущего интервал в один день

Поиск разрывов в версионности

```
SELECT EMP_NAME,
       PREV_END_DATE,
       START_DATE
  FROM (
        SELECT EMP_NAME,
               LAG(END_DATE) OVER (PARTITION BY EMP_NAME
                                   ORDER BY START_DATE) AS PREV_END_DATE,
               START_DATE
          FROM EMP_DEPT
        )
 WHERE START_DATE - PREV_END_DATE > INTERVAL 1 DAY;
```

EMP_NAME	PREV_END_DATE	START_DATE
JONES	1997-11-10	1999-03-05
BLAKE	2002-05-15	2002-05-18

Поиск разрывов в версионности

```
SELECT EMP_NAME,
       PREV_END_DATE + INTERVAL 1 AS MISS_START_DT,
       START_DATE - INTERVAL 1 AS MISS_END_DT
  FROM (
        SELECT EMP_NAME,
               LAG(END_DATE) OVER (PARTITION BY EMP_NAME
                                   ORDER BY START_DATE) AS PREV_END_DATE,
               START_DATE
          FROM EMP_DEPT
        )
 WHERE START_DATE - PREV_END_DATE > INTERVAL 1 DAY;
```

EMP_NAME	MISS_START_DT	MISS_END_DT
JONES	1997-11-11	1999-03-04
BLAKE	2002-05-16	2002-05-17

Поиск пересечений в версионности

EMP_NAME	DEPT_NO	START_DATE	END_DATE
SMITH	1	1995-05-01	2000-06-15
SMITH	2	2000-06-16	9999-12-31
JONES	1	1996-05-10	1999-07-20
JONES	2	1999-03-05	2001-04-25
BLAKE	1	1996-07-20	1997-01-15
BLAKE	2	1997-01-16	2002-05-20
BLAKE	3	2002-05-18	9999-12-31

Поиск пересечений в версионности

EMP_NAME	DEPT_NO	START_DATE	END_DATE
SMITH	1	1995-05-01	2000-06-15
SMITH	2	2000-06-16	9999-12-31
JONES	1	1996-05-10	1999-07-20
JONES	2	1999-03-05	2001-04-25
BLAKE	1	1996-07-20	1997-01-15
BLAKE	2	1997-01-16	2002-05-20
BLAKE	3	2002-05-18	9999-12-31

В таблице присутствуют 2 пересечения:

1. Сотрудник ‘JONES’ работал в двух отделах в период с ‘1999-03-05’ по ‘1999-07-20’
2. Сотрудник ‘BLAKE’ работал в двух отделах в период с ‘2002-05-18’ по ‘2002-05-20’

Пересечения

- Логика исходных данных
  - сотрудник ‘JONES’ участвовал в проектах двух разных отделов с ‘1999-03-05’ по ‘1999-07-20’
- Ошибка ввода данных
  - у сотрудника ‘BLAKE’ неправильно введена дата начала его работы в отделе 3 или дата окончания его работы в отделе 2

Поиск пересечений в версионности

- Использование аналитических функций
- Разбить данные на группы по ключу таблицы:
  - `EMP_NAME`
- Отсортировать данные в группах по дате начала действия интервала:
  - `START_DATE`
- Проверить, не начинается ли период действия текущей записи раньше, чем заканчивается предыдущий интервал

Поиск пересечений в версионности

```
SELECT EMP_NAME,
       PREV_START_DATE,
       PREV_END_DATE,
       START_DATE,
       END_DATE
  FROM (
        SELECT EMP_NAME,
               LAG(START_DATE) OVER (PARTITION BY EMP_NAME
                                     ORDER BY START_DATE) AS PREV_START_DATE,
               LAG(END_DATE) OVER (PARTITION BY EMP_NAME
                                   ORDER BY START_DATE) AS PREV_END_DATE,
               START_DATE,
               END_DATE
          FROM EMP_DEPT
       )
 WHERE START_DATE <= PREV_END_DATE;
```

EMP_NAME	PREV_START_DATE	PREV_END_DATE	START_DATE	END_DATE
JONES	1996-05-10	1999-07-20	1999-03-05	2001-04-25
BLAKE	1997-01-16	2002-05-20	2002-05-18	9999-12-31

Поиск пересечений в версионности

- Можно решить эту задачу без аналитический функций простым соединением версионных таблиц
- Условия использования такого решения:
  - В качестве соединяемых таблиц будет выступать одна и та же таблица
  - Наличие возможности отличить любые две строки таблицы
  - Нет строк с одинаковым ключом и периодом действия

Поиск пересечений в версионности

```
SELECT E1.EMP_NAME,
       E1.START_DATE AS PREV_START_DATE,
       E1.END_DATE AS PREV_END_DATE,
       E2.START_DATE,
       E2.END_DATE
  FROM EMP_DEPT E1
 INNER JOIN EMP_DEPT E2
    ON E1.EMP_NAME = E2.EMP_NAME
   AND E2.START_DATE BETWEEN E1.START_DATE AND E1.END_DATE
   AND E1.END_DATE <> E2.END_DATE;
```

EMP_NAME	PREV_START_DATE	PREV_END_DATE	START_DATE	END_DATE
JONES	1996-05-10	1999-07-20	1999-03-05	2001-04-25
BLAKE	1997-01-16	2002-05-20	2002-05-18	9999-12-31

Сравнение таблиц

- Нередко возникает необходимость сравнения двух вариантов одной и той же таблицы и построения ***таблицы с различиями (diff-таблицы)***

Сравнение таблиц

EMP

EMP_NAME	JOB	HIRE_DATE	SALARY
ADAMS	CLERK	1987-06-23	1100
ALLEN	SALESMAN	1981-02-20	2000
BLAKE	MANAGER	1981-05-01	2850
CLARK	MANAGER	1981-06-09	2450
JAMES	CLERK	1981-12-03	950
JONES	MANAGER	1981-04-02	2975

EMP_OLD

EMP_NAME	JOB	HIRE_DATE	SALARY
ADAMS	CLERK	1987-05-23	1100
ALLEN	SALESMAN	1981-02-20	1600
BLAKE	MANAGER	1981-05-01	2850
FORD	ANALYST	1981-12-03	3000
JAMES	CLERK	1981-12-03	950

Хотим сравнить таблицы EMP_OLD (до внесения изменений пользователей), EMP – после.
Были внесены следующие изменения:

- Добавлены сотрудники ‘CLARK’ и ‘JONES’ (операции INSERT)
- Удален сотрудник ‘FORD’ (операция DELETE)
- Изменена дата рождения сотрудника ‘ADAMS’ и зарплата сотрудника ‘ALLEN’ (операции UPDATE).

Сравнение таблиц

- Таблица строк `EMP`, отсутствующих в `EMP_OLD`
  - результаты выполнения операций `INSERT` и `UPDATE`
- Таблица изменений неключевых полей
  - результаты работы операций `UPDATE`
- Таблица изменений в ключевых полях
  - результаты работы операций `INSERT` и `DELETE`
- Таблица всех изменений

Сравнение таблиц

- Таблица строк EMP, отсутствующих в EMP_OLD

```
SELECT EMP_NAME, JOB, HIRE_DATE, SALARY
  FROM EMP
EXCEPT
SELECT EMP_NAME, JOB, HIRE_DATE, SALARY
  FROM EMP_OLD;
```

EMP_NAME	JOB	HIRE_DATE	SALARY
ADAMS	CLERK	1987-06-23	1100
ALLEN	SALESMAN	1981-02-20	2000
CLARK	MANAGER	1981-06-09	2450
JONES	MANAGER	1981-04-02	2975

Сравнение таблиц

- Таблица изменений неключевых полей

```
SELECT EMP.EMP_NAME,
       EMP_OLD.JOB AS OLD_JOB,
       EMP.JOB AS NEW_JOB,
       EMP_OLD.HIRE_DATE AS OLD_HIRE_DATE,
       EMP.HIRE_DATE AS NEW_HIRE_DATE,
       EMP_OLD.SALARY AS OLD_SALARY,
       EMP.SALARY AS NEW_SALARY
  FROM EMP
 INNER JOIN EMP_OLD
    ON EMP.EMP_NAME = EMP_OLD.EMP_NAME
   AND (EMP.JOB <> EMP_OLD.JOB
        OR EMP.HIRE_DATE <> EMP_OLD.HIRE_DATE
        OR EMP.SALARY <> EMP_OLD.SALARY);
```

EMP_NAME	OLD_JOB	NEW_JOB	OLD_HIRE_DATE	NEW_HIRE_DATE	OLD_SALARY	NEW_SALARY
ADAMS	CLERK	CLERK	1987-05-23	1987-06-23	1100	1100
ALLEN	SALESMAN	SALESMAN	1981-02-20	1981-02-20	1600	2000

Сравнение таблиц

- Таблица изменений в ключевых полях

```
SELECT COALESCE(EMP.EMP_NAME, EMP_OLD.EMP_NAME) AS EMP_NAME,
       COALESCE(EMP.JOB, EMP_OLD.JOB) AS JOB,
       COALESCE(EMP.HIRE_DATE, EMP_OLD.HIRE_DATE) AS HIRE_DATE,
       COALESCE(EMP.SALARY, EMP_OLD.SALARY) AS SALARY,
       CASE
         WHEN EMP.EMP_NAME IS NULL
         THEN 'DELETE'
         ELSE 'INSERT'
       END AS DML_OPERATION
  FROM EMP
  FULL JOIN EMP_OLD
    ON EMP.EMP_NAME = EMP_OLD.EMP_NAME
 WHERE EMP.EMP_NAME IS NULL
    OR EMP_OLD.EMP_NAME IS NULL;
```

EMP_NAME	JOB	HIRE_DATE	SALARY	DML_OPERATION
CLARK	MANAGER	1981-06-09	2450	INSERT
JONES	MANAGER	1981-04-02	2975	INSERT
FORD	ANALYST	1981-12-03	3000	DELETE

Сравнение таблиц: все изменения

```
SELECT CASE
         WHEN EMP.EMP_NAME IS NULL
         THEN 'DELETE'
         WHEN EMP_OLD.EMP_NAME IS NULL
         THEN 'INSERT'
         ELSE 'UPDATE'
       END AS DML_OPERATION,
       COALESCE(EMP.EMP_NAME, EMP_OLD.EMP_NAME) AS EMP_NAME,
       EMP_OLD.JOB AS OLD_JOB,
       EMP.JOB AS NEW_JOB,
       EMP_OLD.HIRE_DATE AS OLD_HIRE_DATE,
       EMP.HIRE_DATE AS NEW_HIRE_DATE,
       EMP_OLD.SALARY AS OLD_SALARY,
       EMP.SALARY AS NEW_SALARY
  FROM EMP
  FULL JOIN EMP_OLD
    ON EMP.EMP_NAME = EMP_OLD.EMP_NAME
 WHERE EMP.EMP_NAME IS NULL
    OR EMP_OLD.EMP_NAME IS NULL
    OR EMP.JOB <> EMP_OLD.JOB
    OR EMP.HIRE_DATE <> EMP_OLD.HIRE_DATE
    OR EMP.SALARY <> EMP_OLD.SALARY;
```

Сравнение таблиц: все изменения

DML_OPERATION	EMP_NAME	OLD_JOB	NEW_JOB	OLD_HIRE_DATE	NEW_HIRE_DATE	OLD_SALARY	NEW_SALARY
UPDATE	ADAMS	CLERK	CLERK	1987-05-23	1987-06-23	1100	1100
UPDATE	ALLEN	SALESMAN	SALESMAN	1981-02-20	1981-02-20	1600	2000
INSERT	CLARK		MANAGER		1981-06-09		2450
INSERT	JONES		MANAGER		1981-04-02		2975
DELETE	FORD	ANALYST		1981-12-03		3000	