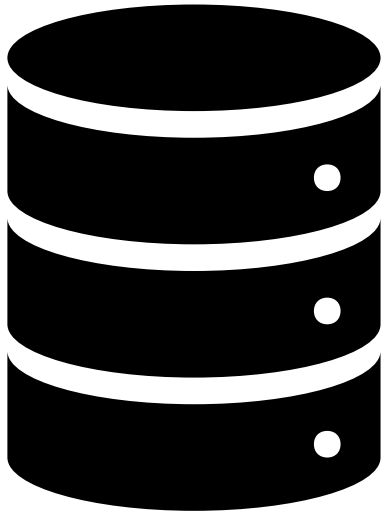

Базы данных

Лекция 5. Проектирование базы данных. Подзапросы.

Меркурьева Надежда
✉ merkurievanad@gmail.com
@merkurievanad

ФПМИ МФТИ, 2021

ОСНОВНЫЕ ШАГИ ПРОЕКТИРОВАНИЯ

- Определение предметной области
- Выделение основных сущностей
- Определение взаимосвязей между сущностями
- Наложение логической структуры на данные
- Создание объектов в базе

ОСНОВНЫЕ ЭТАПЫ ПРОЕКТИРОВАНИЯ

- Концептуальное (инфологическое) проектирование:
  - Итог: концептуальная модель в ER-нотации
- Логическое (даталогическое) проектирование:
  - Итог: логическая модель в ER-нотации
- Физическое проектирование
  - Итог: созданные в базе объекты с учетом их взаимосвязей

КОНЦЕПТУАЛЬНОЕ ПРОЕКТИРОВАНИЕ

- Выделение сущностей
- Определение взаимосвязей между сущностями
- Уточнение типа связей:
  - Один к одному
  - Один ко многим
  - Многие ко многим
- Построение концептуальной модели 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 иллюстрацию 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 PK такой, что PK и FK совпадают с точностью до переименования атрибутов
- В любой момент времени каждое значение FK в текущем значении R_2 идентично значению PK в некотором кортеже в текущем значении R_1. Иными словами, в любой момент времени множество всех значений FK в R_2 является подмножеством значений PK 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 использование 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 таблицы

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,
   ORDER_NUMBER      INTEGER NOT NULL,
   PERSON_ID         INTEGER,
   PRIMARY KEY (ORDER_ID),
   CONSTRAINT FK_PersonOrder FOREIGN KEY (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 PRIMARY KEY,
   ORDER_NUMBER      INTEGER NOT NULL,
   PERSON_ID         INTEGER
                          REFERENCES PERSON(PERSON_ID)
);


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

ССЫЛОЧНАЯ ЦЕЛОСТНОСТЬ

– это необходимое качество реляционной базы данных, заключающееся в отсутствии в любом её отношении внешних ключей, ссылающихся на несуществующие кортежи

- База данных обладает свойством ссылочной целостности, когда для любой пары связанных внешним ключом отношений в ней условие ссылочной целостности выполняется

ПОДДЕРЖАНИЕ ССЫЛОЧНОЙ ЦЕЛОСТНОСТИ

- **CASCADE**
  - При удалении / изменении строки главной таблицы соответствующая запись дочерней таблицы также будет удалена / изменена
- **RESTRICT**
  - Строка не может быть удалена / изменена, если на нее имеется ссылка
  - Значение не может быть удалено / изменено, если на него есть ссылка
- **NO ACTION**
  - Похож на RESTRICT, только проверка происходит в конце транзакции
  - Для разницы с RESTRICT нужно явно прописывать в транзакции выражение
    `SET CONSTRAINTS { ALL | name [, ...] } DEFERRED;`
- **SET NULL**
  - При удалении записи главной таблицы, соответствующее значение дочерней таблицы становится NULL
- **SET DEFAULT**
  - Аналогично SET NULL, только вместо значения NULL устанавливается некоторое значение по умолчанию

FOREIGN KEY

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

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 загуглить 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 ▲ 1	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

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_na... 1	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 ▲ 1	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

КАК ПОСМОТРЕТЬ ОГРАНИЧЕНИЯ В БАЗЕ?

Информация по ограничениям с типом DEFAULT и NOT NULL

```
SELECT *
   FROM information_schema.columns;
```

table_catalog	table_schema	table_name	column_name	column_default	is_nullable	data_type
postgres	taxopark	order	accepted_flg	false	NO	boolean
postgres	taxopark	order_accepting	accepted_flg	false	NO	boolean
postgres	taxopark	trip_payment	payment_type	'CASH'::character varying	NO	character varying
postgres	taxopark	trip_payment	tips_pct	0	NO	double precision
postgres	codeforces	user	contribution	0	NO	integer
postgres	codeforces	contest_x_user	rating_change	0	NO	integer
postgres	codeforces	group	visibility	'Public'::character varying	NO	character varying
postgres	shop	check	order_dttm	(now())::timestamp without time zone	NO	timestamp without tim
postgres	shop	check_x_product	product_cnt	1	NO	integer
postgres	codeforces	submission	end_dt	'2999-01-01'::date	NO	date
postgres	enrollee	university	university_id	nextval('enrollee.university_unive…	NO	integer

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

EMPLOYEE_NM	POSITION_ID	DEPT_ID	VALID_FROM_DTTM	VALID_TO_DTTM
Николай	21	2	2010-08-11 00:00:00	2016-06-06 23:59:59
Николай	23	3	2015-06-07 00:00:00	5999-01-01 00:00:00
Денис	23	3	2010-08-11 00:00:00	2016-06-01 23:59:59
Борис	26	2	2010-08-11 00:00:00	5999-01-01 00:00:00
Пенни	25	2	2010-08-11 00:00:00	5999-01-01 00:00:00

SCD: тип 2

- В полях `valid_from_dttm` и `valid_to_dttm` обычно не используются значения `NULL`
- Вместо NULL используется некоторая константа, например, `'5999-01-01 00:00:00'` для `valid_to_dttm`, как в примере
- Такой подход упрощает написание условий:

```
WHERE day_dt BETWEEN valid_from_dttm AND valid_to_dttm
```

вместо

```
WHERE day_dt >= valid_from_dttm
  AND (day_dt <= valid_to_dttm
       OR valid_to_dttm 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_DTTM	PREV_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

EMPLOYEE_NM	POSITION_ID	DEPT_ID
Коля	21	2
Денис	23	3
Борис	26	2
Пенни	25	2

Таблица с историей

EMPLOYEE_NM	POSITION_ID	DEPT_ID	HISTORY_DTTM
Коля	21	1	11.08.2010 14:12:05
Денис	23	2	19.12.2012 09:54:57
Борис	26	1	09.01.2018 22:22:22

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?

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 grtst_value
     , least(value_1, value_2)    AS lst_value
  FROM t;
```

GREATEST / LEAST

VALUE_1	VALUE_2	GRTST_VALUE	LST_VALUE
10	15	15	10
10	NULL	10	10
28	13	28	13
NULL	16	16	16
NULL	NULL	NULL	NULL

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;
```

СОЕДИНЕНИЕ ВЕРСИОННЫХ ТАБЛИЦ

```
SELECT emp_dept.emd_name
     , dept.dept_name
     , greatest(emp_dept.start_date, dept.start_date) AS start_date
     , least(emp_dept.end_date, dept.end_date) 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;
```

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 самодостаточными 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 1+ СТРОК

- Иногда 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