ifis

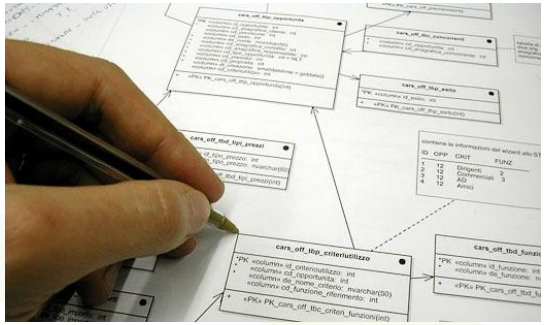
Institut für Informationssysteme
Technische Universität Braunschweig

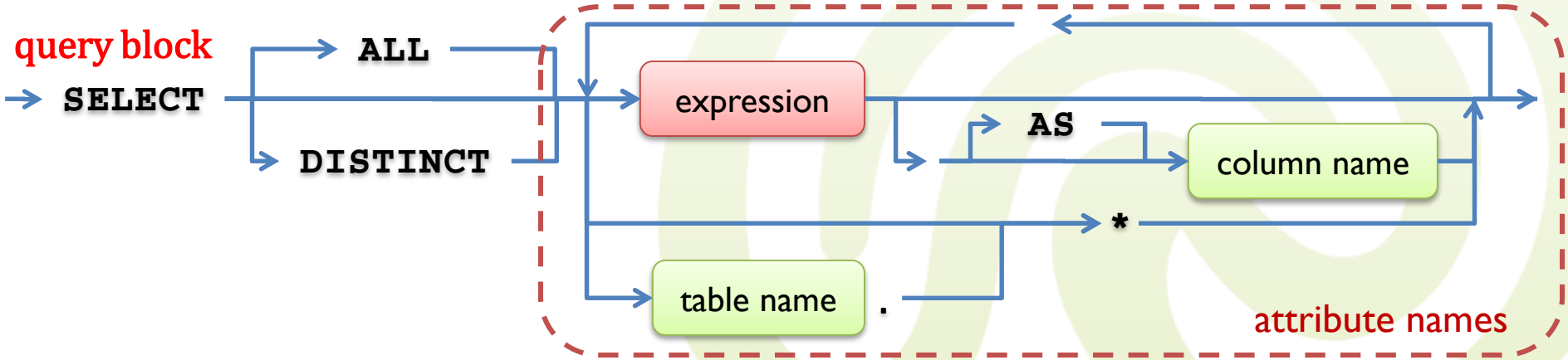
Relational Database Systems I

Wolf-Tilo Balke
Simon Barthel
Institut für Informationssysteme
Technische Universität Braunschweig
www.ifis.cs.tu-bs.de

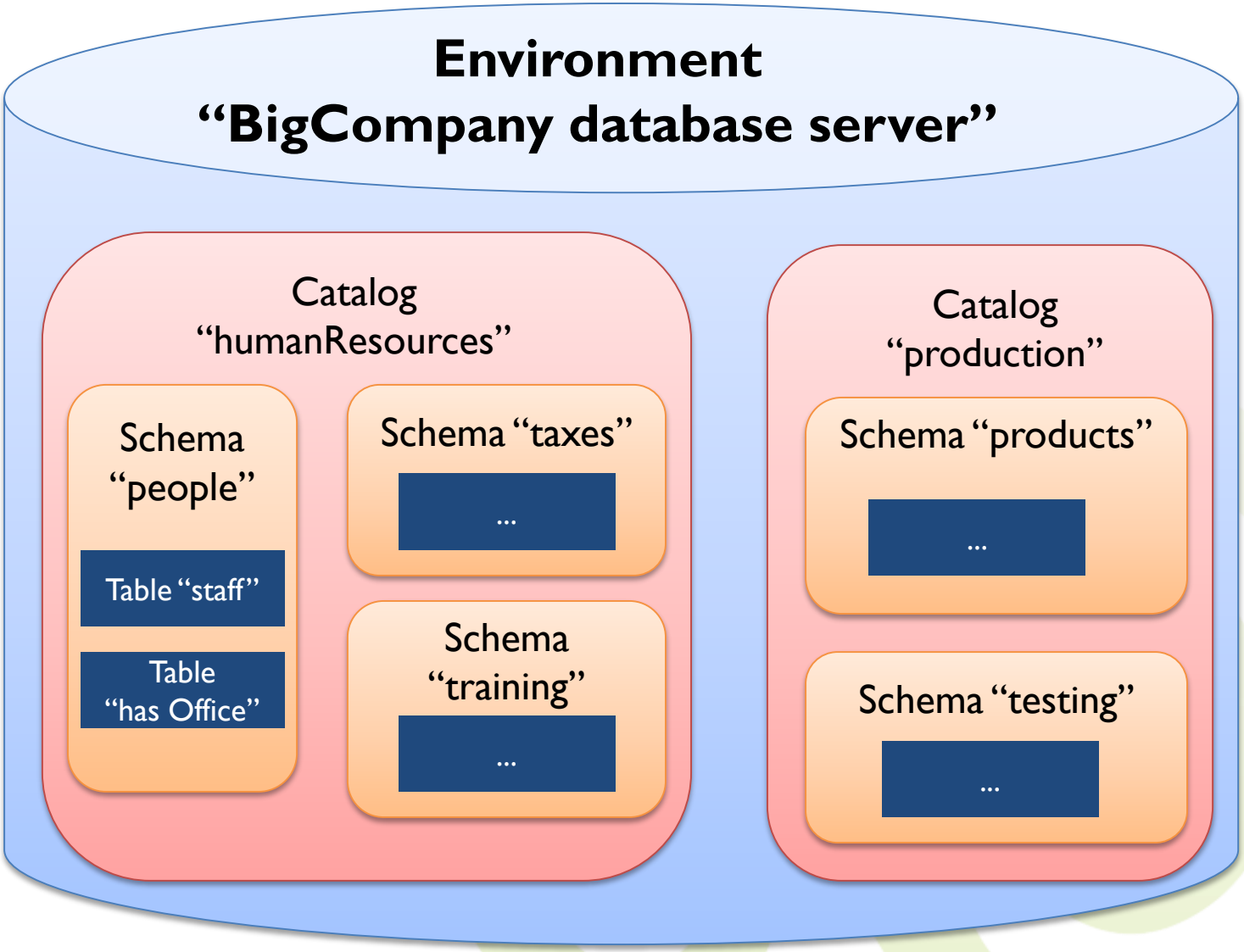
Overview

- SQL data definition language
- SQL data manipulation language (apart from **SELECT**)
- SQL ≠ SQL
- Some advanced SQL concepts





9.1 Recap


Environment
“BigCompany database server”
Catalog
“humanResources”
Schema
“people”
Table “staff”
Table
“has Office”
Schema “taxes”
...
Schema
“training”
...
Catalog
“production”
Schema “products”
...
Schema “testing”
...

9.1 SQL DDL

- Last week, you learned how to query an existing relational database ...
- What's missing?
  - How to **create** schemas, tables, ...
  - How to **drop** schemas, tables, ...
  - How to **alter** schemas, tables, ...

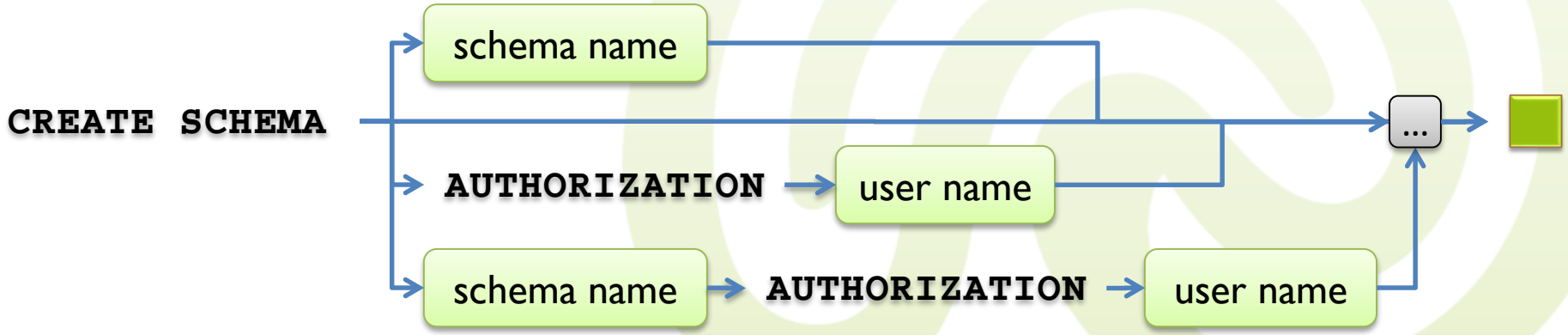
DDL

  - How to **insert** new tuples into existing tables?
  - How to **delete** tuples from existing tables?
  - How to **update** tuples in existing tables?

DML

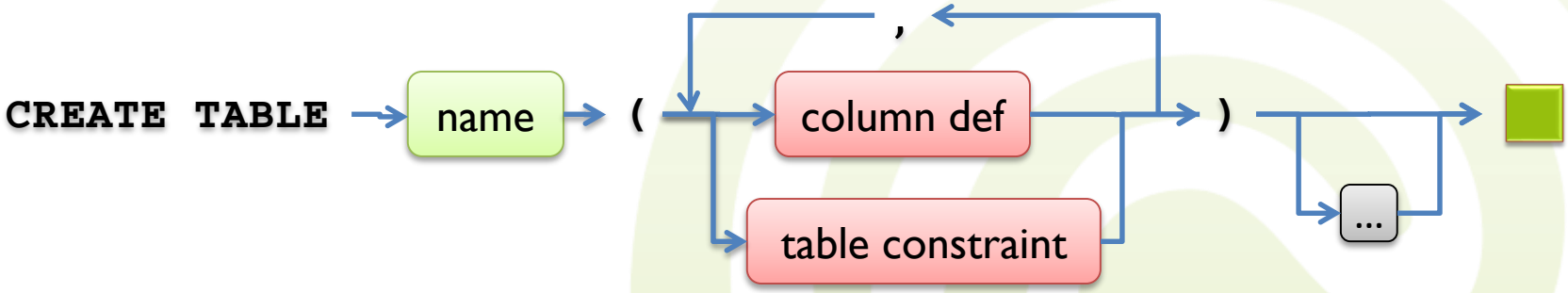
9.1 SQL DDL: Schemas

- **`CREATE SCHEMA`** creates a **new schema** with a given name for a given **owner**
  - If no schema name is provided, the **current username** is used
  - If no explicit owner is provided, also the **current user** is used
- Example
  - `CREATE SCHEMA heroes AUTHORIZATION barthel`



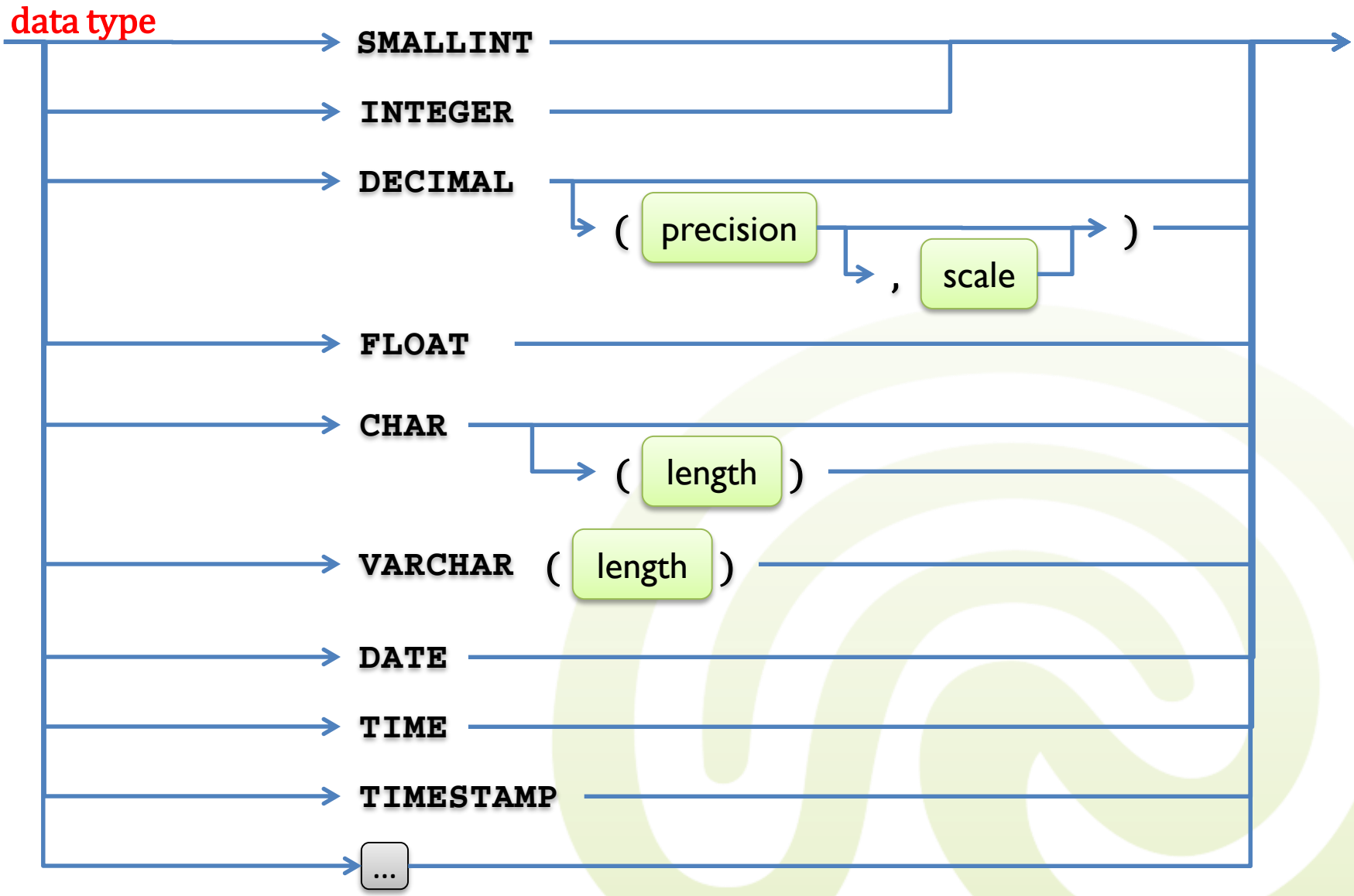
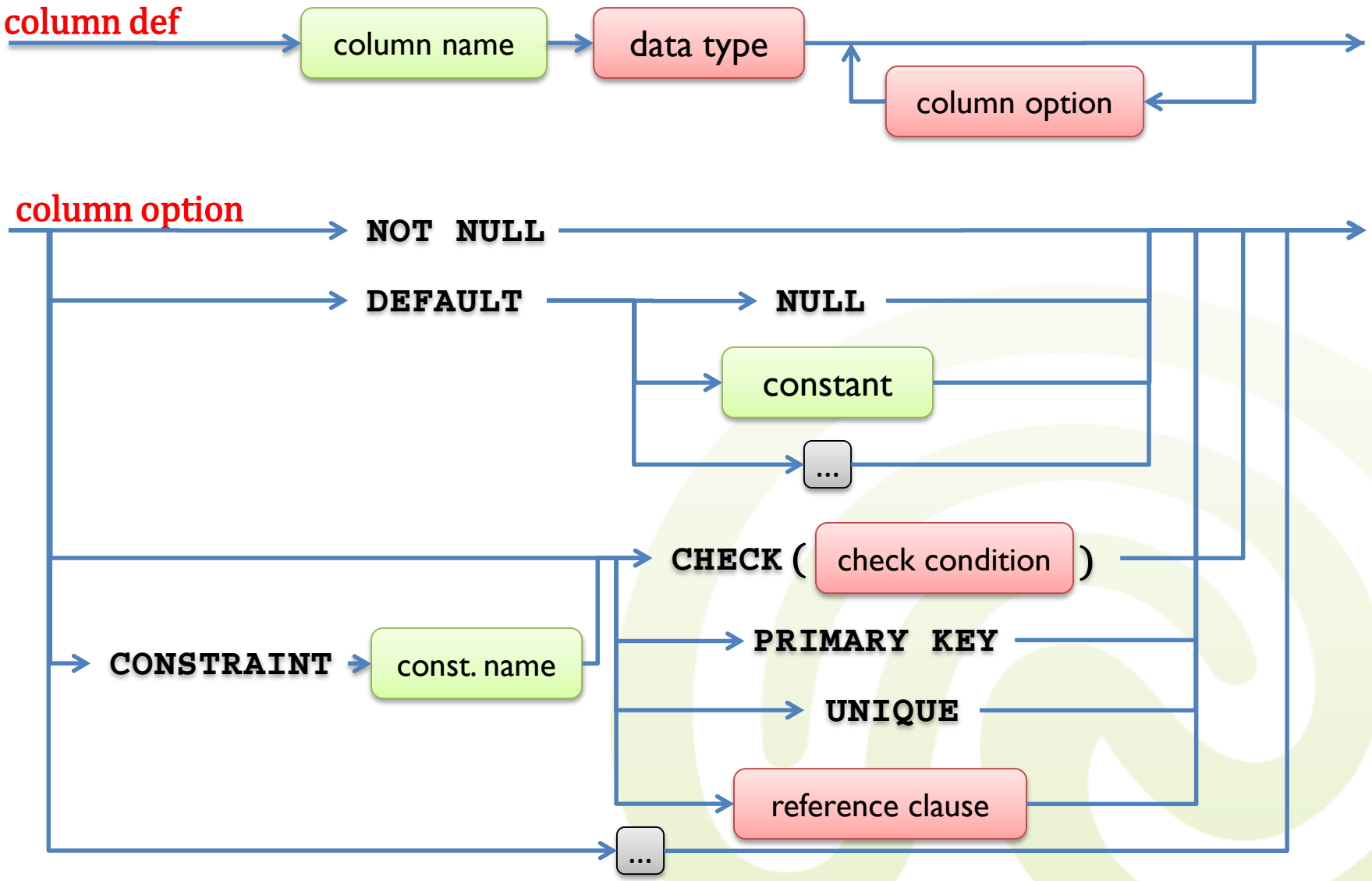
9.1 SQL DDL: Tables

- **`CREATE TABLE`** creates a **new table** with a given name
  - Contains column definition for **each column**
  - Contains additional table-specific **structural constraints**



9.1 SQL DDL: Tables

- Each column has a **name** and a **data type**
- Each column may have multiple column **options**
- Example:
  - ```
    CREATE TABLE person (
        name VARCHAR(200),
        age INTEGER
    )
    ```

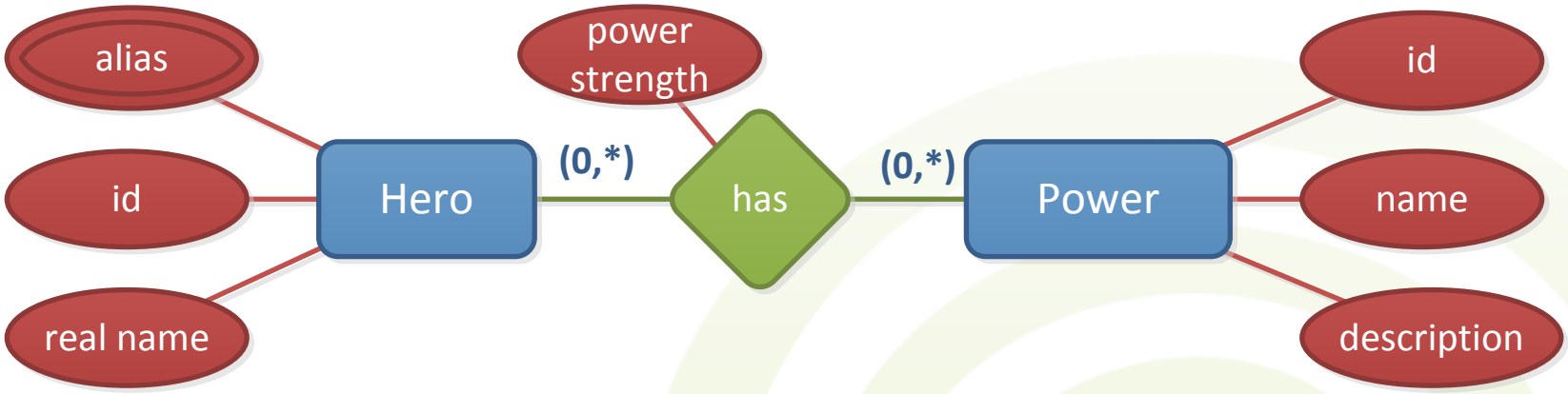
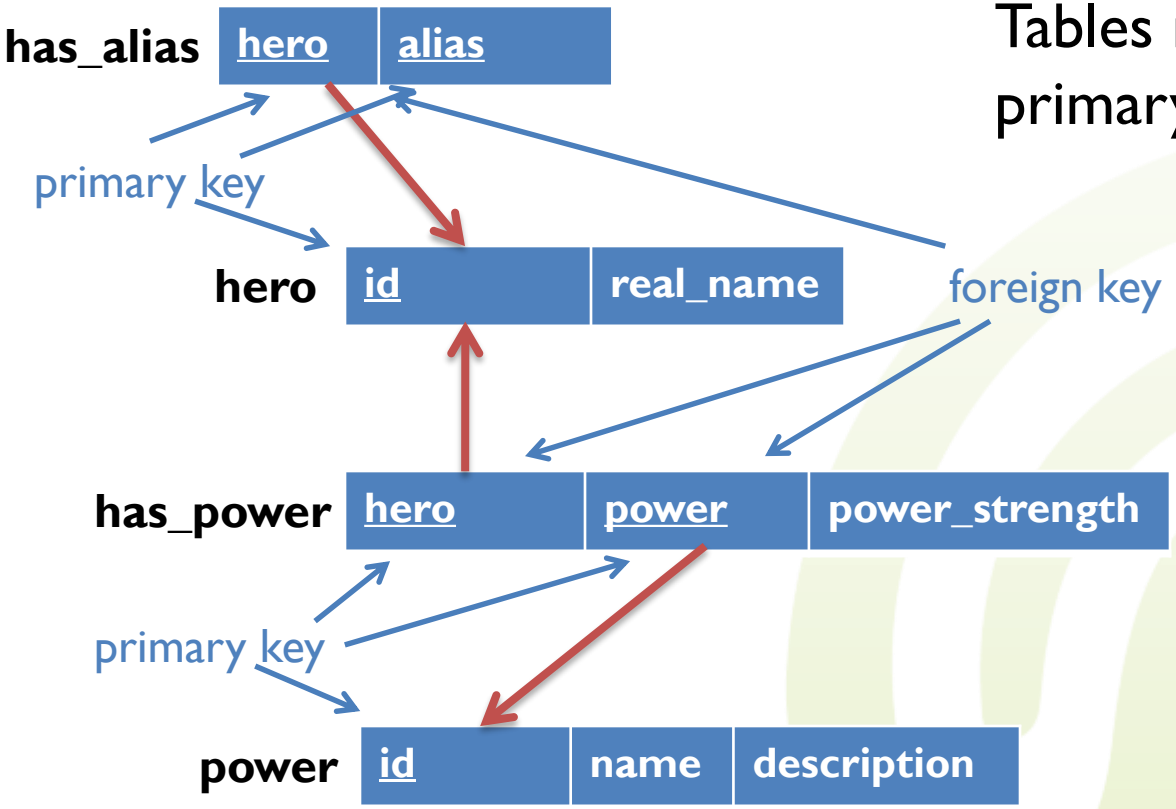
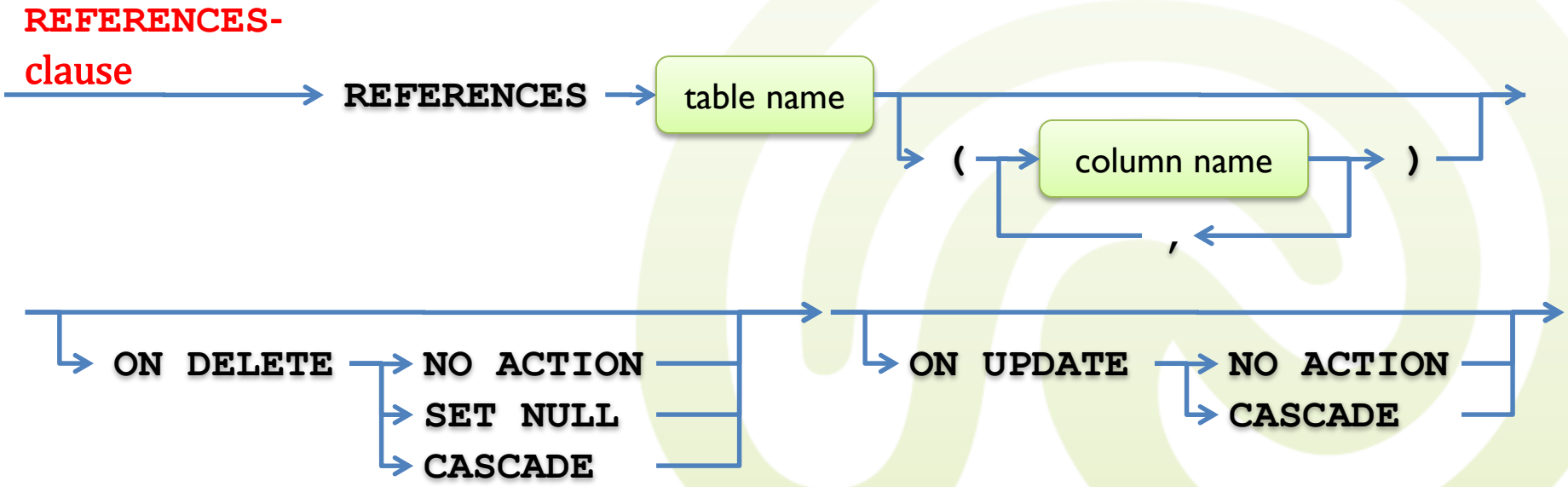
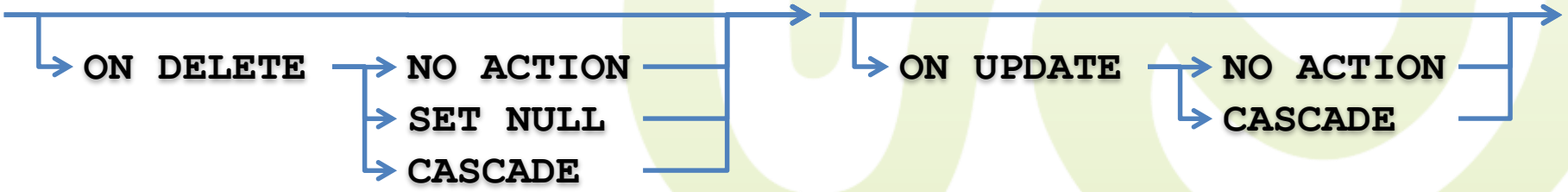
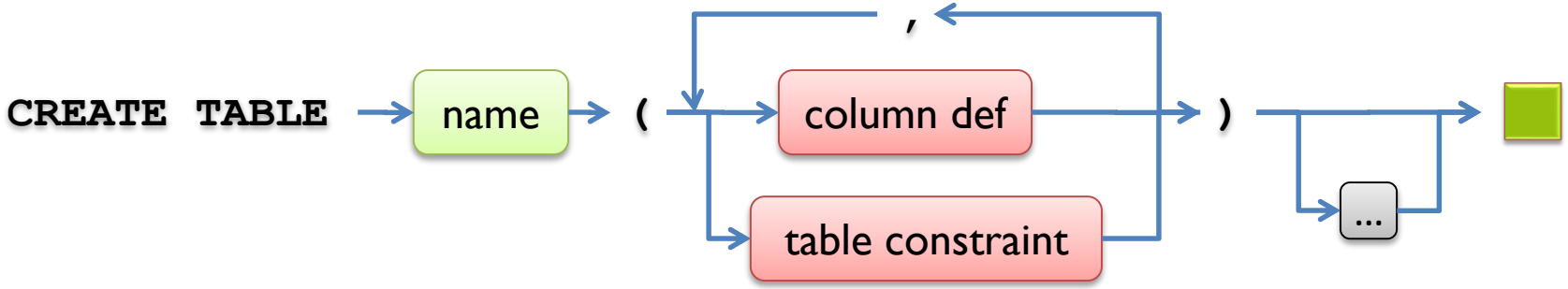
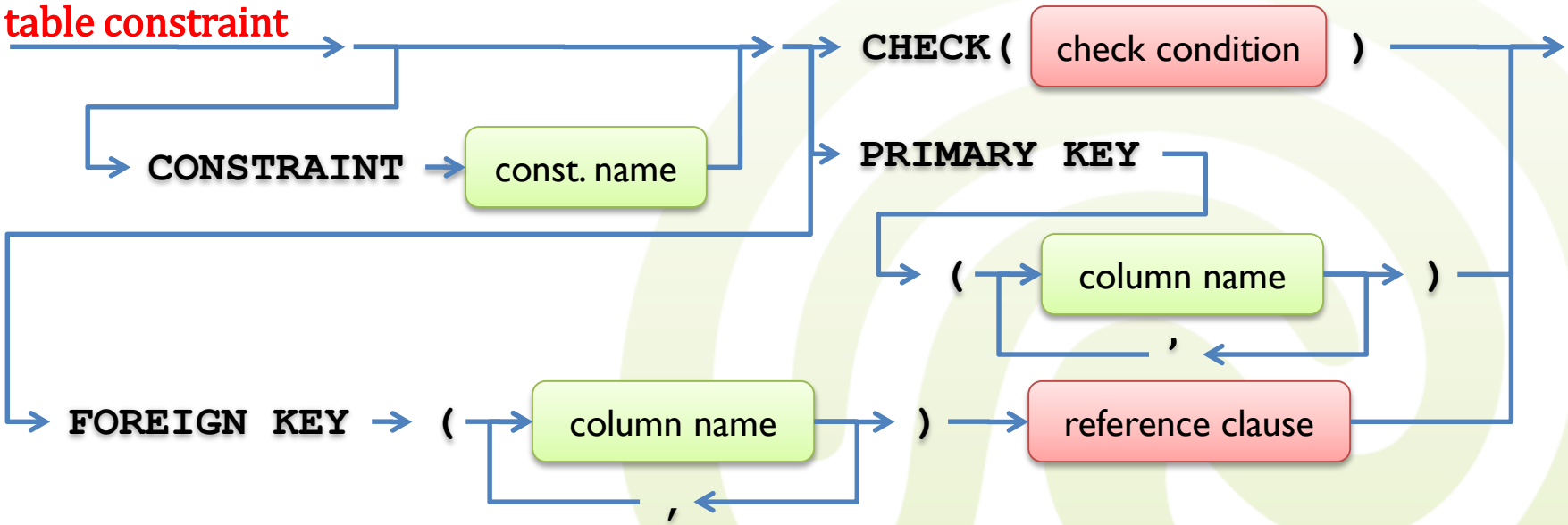
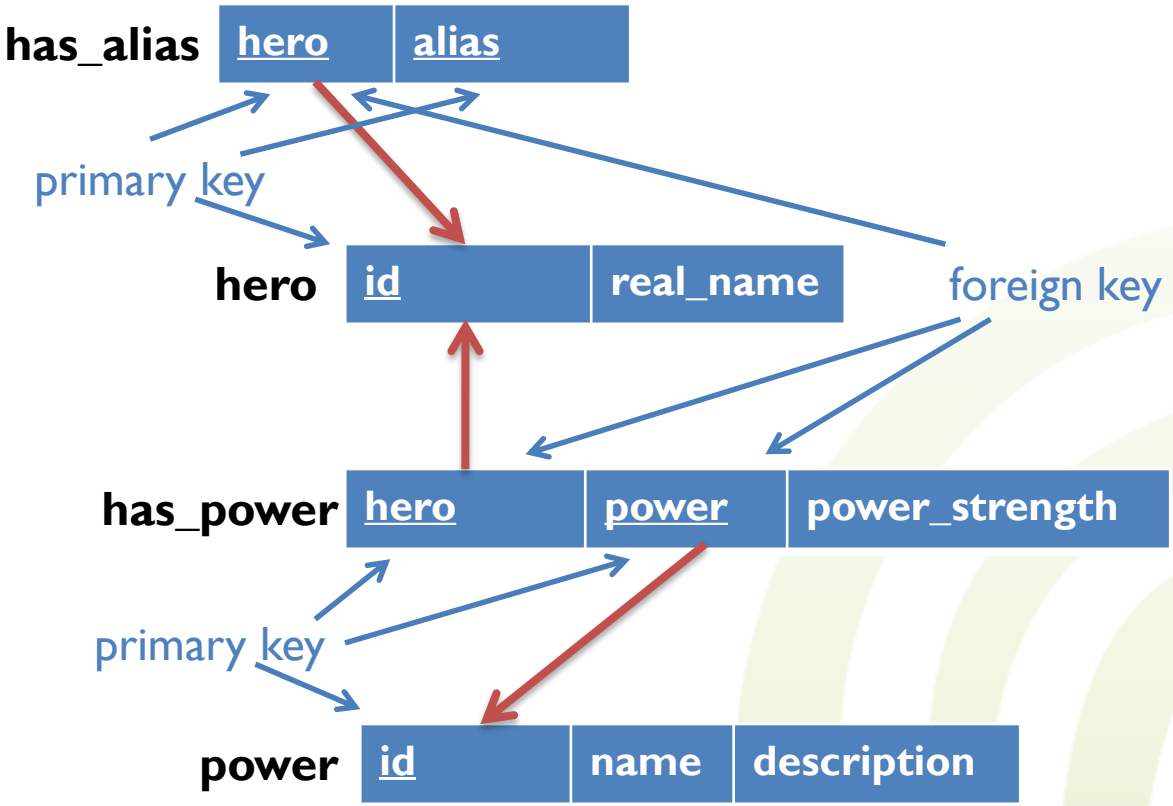
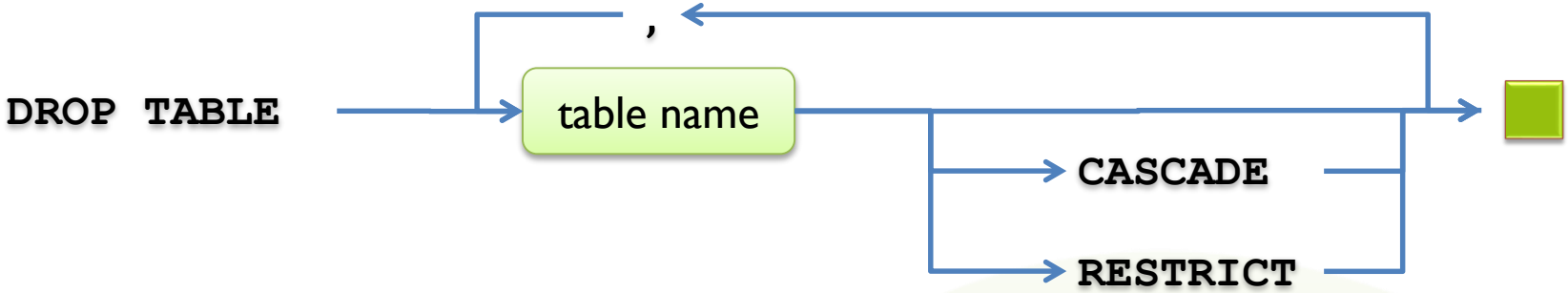
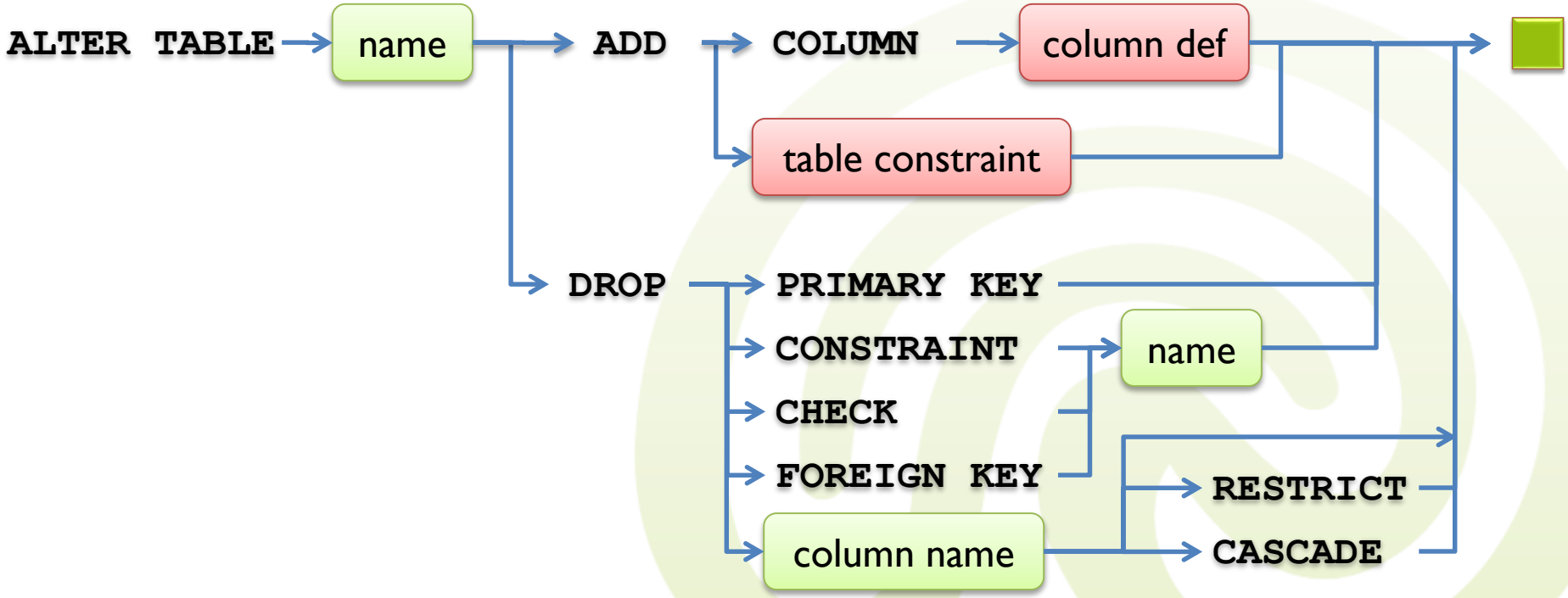

    ```

9.1 SQL DDL: Tables



9.1 SQL DDL: Tables

Name	Syntax	description
Integer	`INTEGER`	Integer value
Float	`FLOAT`	Floating point number of approximate precision (the supported precision is implementation-dependent)
Numeric	`NUMERIC(p,s)`	A number with *p* digits before the decimal point and *s* digits after the decimal point
Character	`CHAR(x)`	A textual string of length *x*
Character varying	`VARCHAR(x)`	A textual string of length at most *x*
Date	`DATE`	Year, month, and day
Time	`TIME`	Hours, minutes, and seconds
Timestamp	`TIMESTAMP`	A date and a time

9.1 SQL DDL: Tables



9.1 SQL DDL: Tables

- **`NOT NULL`:**
  The **`NULL`** value is not allowed for the column
- **Example:**
  - ```
    CREATE TABLE person (
          name VARCHAR(200) NOT NULL,
          age INTEGER NOT NULL
    )
    ```

- **`DEFAULT`:**
  Defines the default value if a value is not explicitly set
  - Usually a constant or **`NULL`**
  - If omitted, **`NULL`** is the default
    ```

9.1 SQL DDL: Tables

- **Column constraints:**
  - Restricts possible values for the current column
  - May have a **unique name** indicated by **`CONSTRAINT`** <name>
    - If name is omitted, system creates a default name
  - **`CHECK`**: User-defined constraint.
    To be valid, values have to satisfy the condition.
  - **Example:**

```
CREATE TABLE person (
        name VARCHAR(200),
        age INTEGER CONSTRAINT adult
            CHECK (age >= 18)
)
```

9.1 SQL DDL: Tables

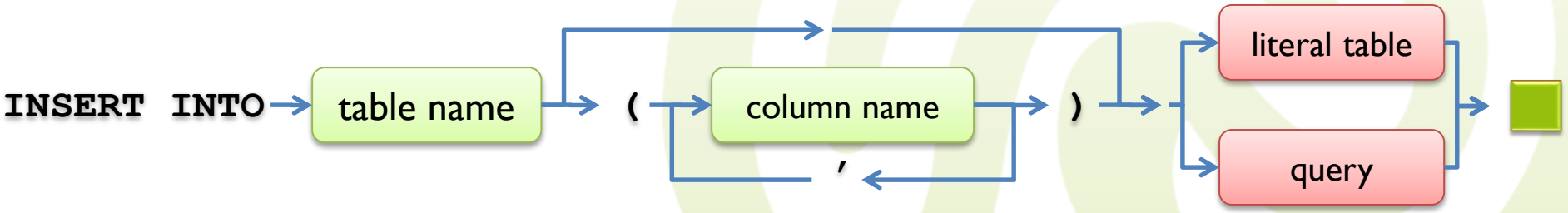
- **UNIQUE**: No duplicate values are allowed within this attribute
  - This option can only be used if the uniqueness constraints concerns only a single attribute
  - For multi-attribute uniqueness constraints, there is a different option (later)
  - Implies **NOT NULL**
    - Note: In DB2, **NOT NULL** has to be specified nevertheless ...
- **Example:**
  - ```
    CREATE TABLE person (
          name VARCHAR(200) NOT NULL UNIQUE,
          age INTEGER NOT NULL
    )
    ```
    ```

9.1 SQL DDL: Tables

- **`PRIMARY KEY`**: Each table may have a **primary key** (optionally, but recommended) made up of at least one column
  - This option can only be used if the primary key consists of only one column
  - For multi-column primary keys, you need a different option (later)
  - Implies **`NOT NULL`** and **`UNIQUE`**
    - Again, DB2 needs an explicit **`NOT NULL`**
- Additionally, a **referential clause** may be specified (see next slides)

9.1 SQL DDL: Referential Integrity

- Rows in tables may **refer** to rows in other tables to capture **relationships**
- Of course, you should not be allowed to refer to a non-existing row
  - **Referential integrity** between **primary keys** and **foreign keys** ensures that references are correct

9.1 SQL DDL: Referential Integrity

Example:

Conceptual ER schema:



9.1 SQL DDL: Referential Integrity

Resulting tables:



Tables refer to others by primary keys and foreign keys

9.1 SQL DDL: Referential Integrity

- Referential integrity can be defined using the **`REFERENCES` clause**
  - Either used by constraints in **column options** or within **table constraints**



9.1 SQL DDL: Referential Integrity

- **Example:**
  - ```
    CREATE TABLE employee(
        id INTEGER NOT NULL PRIMARY KEY,
        name VARCHAR(100) NOT NULL
    )
    ```
  - ```
    CREATE TABLE managed_by(
        employee INTEGER NOT NULL
            REFERENCES employee,
        manager INTEGER NOT NULL
            REFERENCES employee
    )
    ```

9.1 SQL DDL: Referential Integrity

- Optionally, you may specify what happens if a row that is referenced will be deleted or modified
  - **`ON DELETE`**: If a referenced row is deleted, ...
    - **`NO ACTION`**: ...reject the deletion (that is, it cannot be performed)
    - **`SET NULL`**: ...delete it and set all referencing foreign keys to **`NULL`**
    - **`CASCADE`**: ...delete it along with all rows referring to it
  - **`ON UPDATE`**: If the primary key of a referenced row is modified, ...
    - **`NO ACTION`**: ...reject the modification (that is, it cannot be performed)
    - **`CASCADE`**: ...change all values of referencing foreign keys
  - Default:
    - **`ON DELETE NO ACTION ON UPDATE NO ACTION`**



9.1 SQL DDL: Table Constraints





9.1 SQL DDL: Table Constraints

- Table constraints behave similar to constraints in column options
  - If no name is provided, a **name is automatically generated**
  - The **`CHECK` condition** may contain any Boolean predicate
  - In contrast to column options, table constraints may declare **primary keys** consisting of **multiple attributes**
  - **Foreign keys** declare references to primary keys of other tables
    - See referential integrity

9.1 SQL DDL: Table Example



9.1 SQL DDL: Table Example

```
CREATE TABLE hero(
    id INTEGER NOT NULL PRIMARY KEY,
    real_name VARCHAR(100)
)
CREATE TABLE power(
    id INTEGER NOT NULL PRIMARY KEY,
    name VARCHAR(100),
    description VARCHAR(255)
)
CREATE TABLE has_alias (
    hero INTEGER REFERENCES hero
        ON DELETE CASCADE
        ON UPDATE CASCADE,
    alias VARCHAR(100) NOT NULL,
    PRIMARY KEY (hero, alias)
)
```

link has_alias to hero

delete alias if hero is deleted

update alias if hero is updated

composed primary key

9.1 SQL DDL: Table Example

```
CREATE TABLE has_power(
        hero INTEGER NOT NULL,
        power INTEGER NOT NULL,
        power_strength INTEGER NOT NULL,
        PRIMARY KEY (hero, power),
        FOREIGN KEY (hero) REFERENCES hero
            ON DELETE CASCADE
            ON UPDATE CASCADE,
        FOREIGN KEY (power) REFERENCES power
            ON DELETE CASCADE
            ON UPDATE CASCADE
)
```

9.1 SQL DDL: Drop Tables

- For **deleting** tables, there is the **`DROP TABLE`** command



  - If **`RESTRICT`** is used, you may only drop empty tables that are not referenced by any other table
  - If **`CASCADE`** is used, all referencing tables are also deleted (including all stored rows)
    - DB2 does not support **`CASCADE`** …
  - If neither is used, the table does not have to be empty, but must not be referenced by another one
  - Example
    - **`DROP TABLE hero CASCADE, power CASCADE`**

9.1 SQL DDL: Alter Tables

- After a table has been created, you may **alter** it by adding/removing **columns** or **constraints**



9.1 SQL DDL: Alter Tables

- If you add a **new column** with a **`NOT NULL`** constraint, you also need to provide a **default value**
- When **dropping** a **column,** you must either choose
  - **`CASCADE`** to also delete any views, indexes, and constraints dependent on that column
  - **`RESTRICT`** to allow the drop only if there is no referring column (default)
- If the **name** of a constraint is **auto-generated**, you need to look it up in the **system catalog**
- **Example:**
  - ```
    ALTER TABLE has_power DROP power_strength
    ALTER TABLE has_power
          ADD COLUMN since DATE
    ```
    ```

9.2 SQL DML

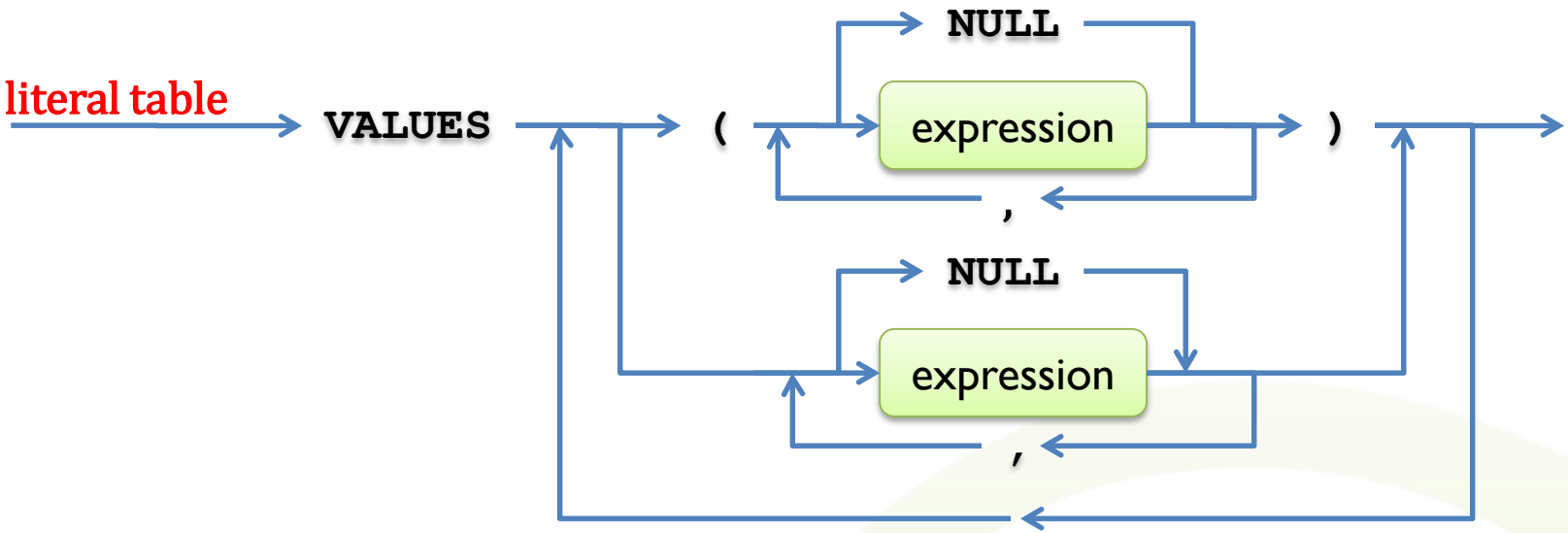
- **Data definition language** (DDL)
  - Creating, changing, altering schemas, tables, ...
    - `CREATE SCHEMA`
    - `CREATE TABLE`
    - `ALTER TABLE`
    - `DROP TABLE`
- **Data manipulation language** (DML)
  - Querying
    - `SELECT`
  - Adding and updating data
    - `INSERT INTO`
    - `UPDATE`
    - `DELETE`

9.2 SQL DML: Insert

- Now we have wonderful, empty tables
- We need to put data into them!
  - **`INSERT INTO`** statement
  - You can specify into what **columns** you want to insert data
    - Default: All columns
  - New values are stated as a **literal table** or **inline view (query)**
    - Of course the **attribute domains** have to match



9.2 SQL DML: Insert



- A **literal table** is defined extensionally:

```
VALUES ('James', 'Howlet')

VALUES ('Charles', 'Xavier'), ('Jean', 'Grey')

VALUES 'Wolverine', ('Professor X'), 'Phoenix'
```

James	Howlet

Charles	Xavier
Jean	Grey

Wolverine
Professor X
Phoenix

9.2 SQL DML: Insert

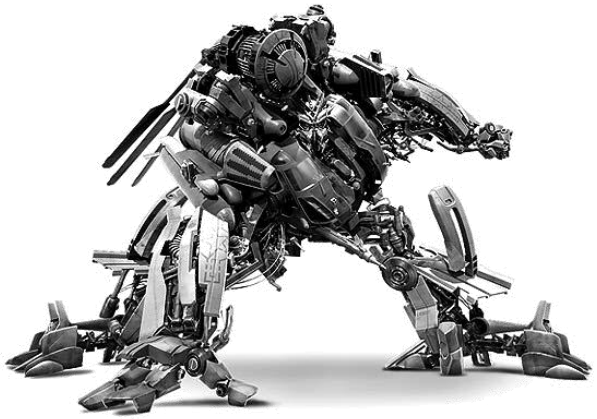
```
INSERT INTO hero(id, real_name) VALUES
     (1, 'Charles F. Xavier'),
     (2, 'Jean Grey')
INSERT INTO has_alias VALUES
     (1, 'Professor X'),
     (1, 'Onslaught'),
     (2, 'Phoenix'),
     (2, 'Marvel Girl')
```

9.2 SQL DML: Insert

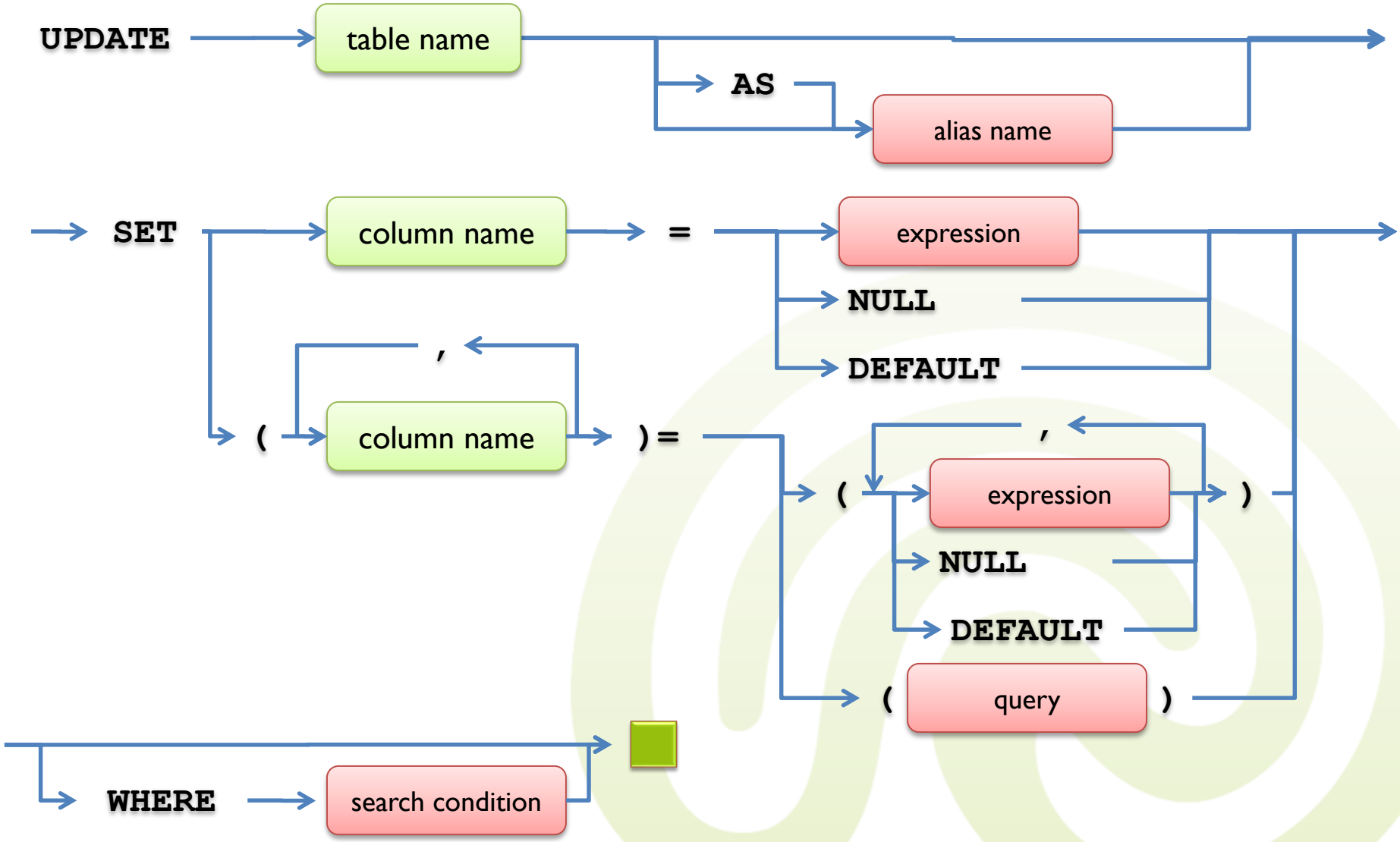
- Of course, subqueries may also be used in **INSERT** statements
  -
```
INSERT INTO heroes_starting_with_a(
    SELECT * FROM hero
    WHERE real_name LIKE 'A%'
)
```

9.2 SQL DML: Update

- Existing rows can also be changed using the **`UPDATE` statement**
  - Very similar to the **`SELECT`** statement
  - Update **finds rows** fulfilling a **given condition** and **changes** some of its **rows** by assigning **new values**



9.2 SQL DML: Update

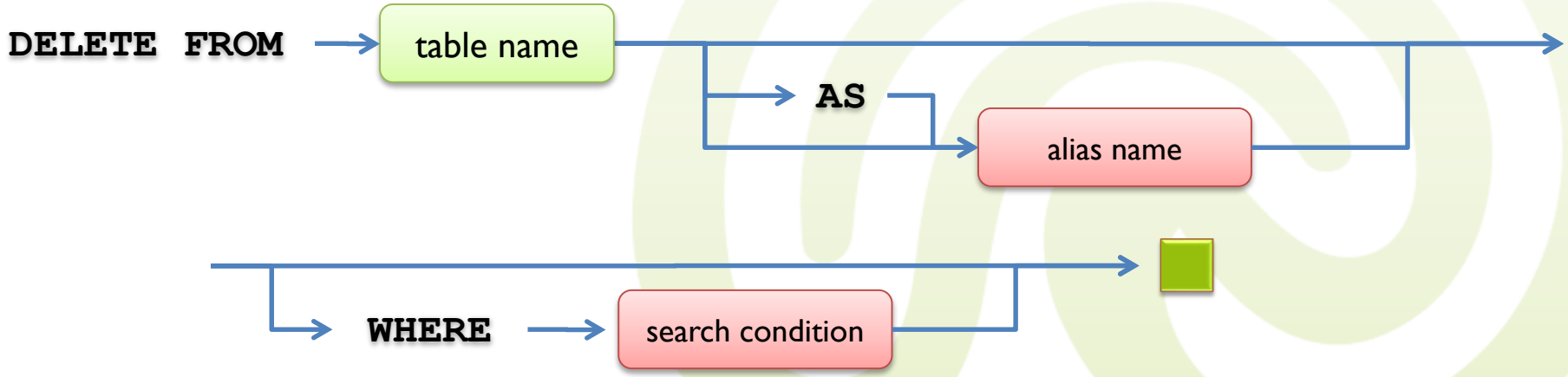


9.2 SQL DML: Update

- Hide the real name of each hero
  - `UPDATE hero SET real_name = NULL`
- Multiply all power_strength values by 10
  - `UPDATE has_power SET power_strength = power_strength * 10`
- Change the name of hero with id 1
  - `UPDATE hero SET name= 'Charles Francis Xavier' WHERE id = 1`
- Change name and id of Jean Grey
  - `UPDATE hero SET (id, name) = ('3', 'Jean Grey-Summers') WHERE name = 'Jean Grey'`
  - Change of id is propagated to other tables when `ON UPDATE CASCADE` is used in table definition
- Again, subqueries can be used in the **`WHERE`** clause

9.2 SQL DML: Delete

- The **DELETE statement** is used to delete rows from a table
  - Deletes all rows satisfying a certain search condition
  - **Example:**
    - Delete Jean Grey
      - `DELETE FROM hero WHERE name = 'Jean Grey'`
    - Delete all heroes
      - `DELETE FROM hero`



9.2 SQL DML: Delete

- Again, subqueries can be used here
  - ```
    DELETE FROM hero h
    WHERE NOT EXISTS(
        SELECT * FROM has_alias a
        WHERE a.hero = h.id
    )
    ```


    ```

9.3 SQL ≠ SQL

Detour

- First, the **good news:**
  - SQL has been standardized by the ISO in 1987
  - The standard is well-maintained and under active development
    - SQL-86, SQL-89, **SQL-92, SQL:1999, SQL:2003,** SQL:2006, SQL:2008
  - Many "big" database vendors participate in the standardization process:
    - IBM, Ingres, Microsoft, Oracle, Sybase, ...

ISO International Organization for Standardization

9.3 SQL ≠ SQL

Detour

A timeline of SQL standardization:

- 1986
  - ANSI SQL
  - Relations, attributes, views
  - **`SELECT` … `FROM` … `WHERE` …**
- 1987
  - SQL-86 (ISO/IEC 9075:1986)
- 1989
  - SQL-89 (SQL1)
    - ≈ SQL-86 + restricted referential integrity

9.3 SQL ≠ SQL

Detour

- 1992
  - SQL-92 (SQL2)
  - 3 parts, 1120 pages
  - Entry Level
    - ≈ SQL-89 + **`CHECK`** (attribute)
  - Intermediate Level
    - ⊇ Entry Level + domains, **`CHECK`** (relation), **`CASE`**, **`CAST`**, **`JOIN`**, **`EXCEPT`**, **`INTERSECT`**
  - Full Level
    - ⊇ Intermediate Level + assertions, nested select, nested from

9.3 SQL ≠ SQL

Detour

- 1999/2000
  - SQL:1999 (SQL3)
  - 5 parts, 2084 pages
  - ≈ SQL-92 + object-orientation, **recursive queries,** triggers, OLAP, user-defined types, regular expressions
  - Computationally complete, object-oriented database programming language, descriptive and procedural
  - Core (about180 features)
    - ≈ SQL92 Entry Level + parts of Intermediate and Full Level
  - 9 Packages (about 240 features)
    - enhanced date/time, enhanced integrity, OLAP, PSM, CLI, basic object support, enhanced object support, trigger, SQL/MM

9.3 SQL ≠ SQL

Detour

- Recursive queries (SQL:1999):
  - How to find all rivers that flow into the North Sea?
  - Multiple joins?

flows_into	river	mouth
	Oker	Aller
	Aller	Weser
	Weser	North Sea
	Elbe	North Sea
	Edder	Wietze
	Flöth	Wietze
	Wietze	Aller
	Isar	Danube
	Inn	Danube

9.3 SQL ≠ SQL

Detour

- Solution:

```
WITH RECURSIVE flows_into_ns(river) AS (
    SELECT river
    FROM flows_into
    WHERE mouth = 'North Sea'
        UNION
    SELECT fi.river
    FROM flows_into AS fi
        JOIN flows_into_ns AS fins
            ON fi.mouth = fins.river
)
SELECT river FROM flows_into_ns
```

- Note: DB2 uses a different syntax
  - No **RECURSIVE**, **UNION ALL** instead of **UNION**, no explicit **JOIN**

9.3 SQL ≠ SQL

Detour

- 2003
  - SQL:2003
  - 14 parts, 3606 pages
  - **`MULTISET`** as an explicit construct (with numerous operations, such as: **`MULTISET UNION`**, **`MULTISET EXCEPT`**, **`MULTISET INTERSECT`**, **`CARDINALITY`**)
  - Sequence generators
    - **`CREATE SEQUENCE`** <sequence name> **`AS`** <type name>
      [**`START WITH`** <value>] [**`INCREMENT BY`** <value>]
      [**`NO MINVALUE`** | **`MINVALUE`** <value>]
      [**`NO MAXVALUE`** | **`MAXVALUE`** <value>]
      [**`NO CYCLE`** | **`CYCLE`**]
  - Base type XML for mappings between SQL and XML

9.3 SQL ≠ SQL

Detour

- 2006
  - SQL:2006
  - Successor of SQL:2003
  - A new extension for XML handling
    - Importing, storing, querying, and manipulating XML data
    - Support for XQuery
    - Concurrently access (object-)relational data and XML data
- 2008
  - SQL:2008
  - Successor of SQL:2006
  - “Maintenance release” (some new but minor features)

9.3 SQL ≠ SQL

Detour

- Well, here are the **bad news:**
  - There are still **too many variants** of SQL (both **syntactic and semantic differences**)
    - True application portability remains a challenge
  - The standard has been used to introduce **two kinds of features:**
    1. Features that are well-understood and widely implemented
    2. New and largely untried technologies, hoping that vendors follow the lead and deliver new functionalities
  - **Vendors don't care** too much about the standard

9.3 SQL ≠ SQL

Detour

- A **common myth** among software developers:

If your application uses only standard SQL, then it is portable.

- If you don't believe me, here are some examples ...

9.3 SQL ≠ SQL

Detour

- ```
  CREATE TABLE name (
      first VARCHAR(100),
      middle VARCHAR(100),
      last VARCHAR(100)
  )

  INSERT INTO name VALUES ('George', 'Walker', 'Bush')
  INSERT INTO name VALUES ('Horst', '', 'Kr')
  INSERT INTO name VALUES ('Angela', NULL, 'Merkel')
  ```
  - `''` (empty string) means that we know that there is no middle name
  - **NULL** means that we don't know whether there is a middle name

- Sounds like a good design? What do you think?
  - According to the SQL standard, this approach is fine ...
  - ... unless your RDBMS is Oracle (`''` is the same as **NULL**)
  ```

9.3 SQL ≠ SQL

Detour

- What about terminology?
  - The SQL standard defines the following notions:
    - Environment
    - Cluster
    - Catalog
    - Schema
  - The reality:
    - Database server
    - (unsupported)
    - Database
    - Schema
  - But attention:
    - In MySQL, there are no catalogs, "schema" and "database" are synonyms
    - In Oracle, there is exactly one schema per user; **`CREATE/ALTER SCHEMA`** x <command> executes <command> on all objects located in schema x

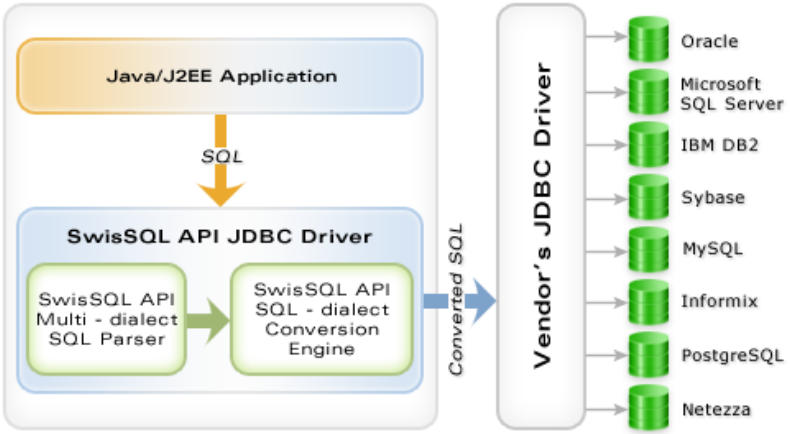
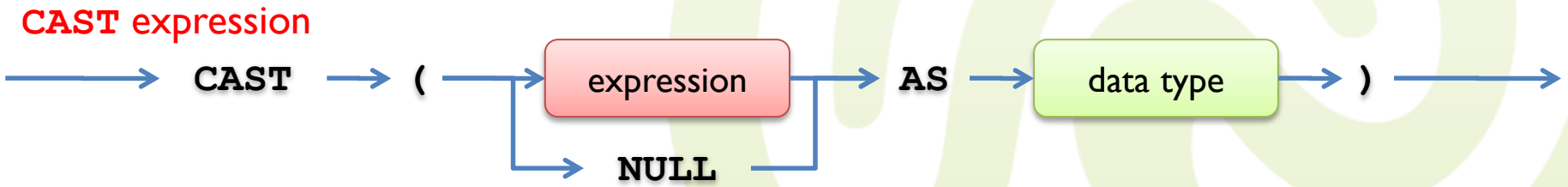
9.3 SQL ≠ SQL

Detour

- The **statement terminator** ";"
  - According to the SQL standard, (almost) every SQL statement has to be terminated by a semicolon
  - What's happening in practice?
    - Many RDBMS treat the terminator as being optional (which is fine, but may cause some problems)
    - Some RDBMS either strictly require a terminator or complain if it is present
    - In some RDBMS, this behavior can be configured ...
  - Summary:
    No matter what you do, it causes problems!

9.3 SQL ≠ SQL

Detour

- The **BOOLEAN data type**
- ```
  CREATE TABLE customers (
      id INTEGER PRIMARY KEY,
      name VARCHAR(100),
      is_vip BOOLEAN,
      is_blacklisted BOOLEAN
  )

  SELECT id, name FROM customers
  WHERE is_vip AND NOT is_blacklisted
  ```
  - Practice?
    - Not supported by Oracle, DB2, and MS SQL Server
      - Official workarounds: Use **CHAR** or **INTEGER** …
    - Supported by MySQL and PostgreSQL
      - Where in MySQL **BOOLEAN** is just a short hand for **TINYINT(1)** …
  ```

9.3 SQL ≠ SQL

Detour

- **Summary**
  - **SQL is not SQL**
  - In some cases, even **identical SQL** code **works differently** on different RDBMS
- Current trends?
  - Open-source RDBMS (PostgreSQL, MySQL, Derby, ...) typically try to adhere to the standard
    - However, many advanced features are not supported yet
  - Recently, DB2 added support for Oracle's SQL (in DB2 9.7)

9.3 SQL ≠ SQL

Detour

- Helpful tools:
  - SQL::Translator (Perl module)
    http://search.cpan.org/dist/SQL-Translator
  - SwisSQL Console (GUI tool, discontinued)
    http://www.swissql.com/products/sql-translator/sql-converter.html
  - SwisSQL API (.NET/Java SQL wrapper, discontinued)
    http://www.swissql.com/products/sqlone-apijava/sqlone-apijava.html
    http://www.swissql.com/products/sqlone-apidotnet/sqlone-apidotnet.html



SwisSQL API(Java) JDBC Wrapper used by Java/J2EE Application

9.4 Advanced Concepts

- Type casting
- Ranking functions
- **`CASE`** expressions

9.4 Type Casting

- SQL is a **strongly typed** language
  - Basically, this means that e.g. `INTEGER` is different from `VARCHAR(100)`
- If data types are incompatible, **type casting** may be used to make them compatible
  - `CAST` expression
  - During casting, precision may be lost (e.g. `FLOAT` → `INTEGER`)
  - Example:
    - ```
      CAST (power_strength AS NUMERIC(3, 2))
      CAST (alias || real_name AS CHAR(255))
      ```


      ```

9.4 Type Casting

- In DB2, possible castings are:

Source	Target
SMALLINT, INTEGER, DECIMAL, FLOAT	SMALLINT, INTEGER, DECIMAL, FLOAT
CHAR, VARCHAR, LONG VARCHAR, CLOB	CHAR, VARCHAR, LONG VARCHAR, CLOB, BLOB
CHAR, VARCHAR	SMALLINT, INTEGER, DECIMAL, DATE, TIME, TIMESTAMP, VARGRAPHICS
SMALLINT, INTEGER, DECIMAL	CHAR
DATE, TIME, TIMESTAMP	CHAR, VARCHAR
DATE	DATE
TIME	TIME
TIMESTAMP	TIMESTAMP
GRAPHICS, VARGRAPHICS, LONG VARGRAPHICS, DBCLOB	GRAPHICS, VARGRAPHICS, LONG VARGRAPHICS, DBCLOB

9.4 Ranking Functions

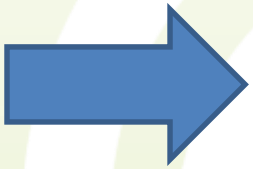
- Since SQL:2003, there are special functions for working with **result lists**
- Examples:
  - Output only every other row of the list
  - Create a ranking with explicit ranks (1, 2, 3, ...)
  - On what rank position is some given row?

9.4 Ranking Functions

- **`ROW_NUMBER()`** returns the **position** of each row in the result list
- **Example:**

```
SELECT name, salary,
  ROW_NUMBER() OVER (
    ORDER BY salary DESC
  ) AS pos
FROM salary
```

person	name	salary
	Christoph	45000
	Wolf-Tilo	75000
	Larry	200000000
	Joachim	45000



name	salary	pos
Larry	200000000	1
Wolf-Tilo	75000	2
Joachim	45000	3
Christoph	45000	4

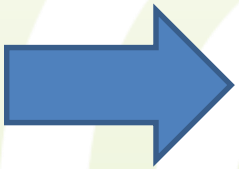
Depending on the implementation, the last two rows may switch positions

9.4 Ranking Functions

– **Example:** At which position is Wolf-Tilo?

```
SELECT name, salary,
   ROW_NUMBER() OVER (
     ORDER BY salary DESC
   ) AS pos
FROM salary
WHERE name = 'Wolf-Tilo'
```

person	name	salary
	Christoph	45000
	Wolf-Tilo	75000
	Larry	200000000
	Joachim	45000



name	salary	pos
Wolf-Tilo	75000	2

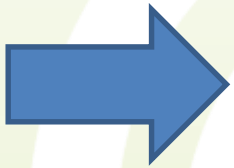
9.4 Ranking Functions

– **Example:** Show only rows at even positions:

```
SELECT name, salary, ROW_NUMBER() OVER (
   ORDER BY salary DESC
) AS pos
FROM salary
WHERE (pos % 2) = 0
```

modulo

person	name	salary
	Christoph	45000
	Wolf-Tilo	75000
	Larry	200000000
	Joachim	45000



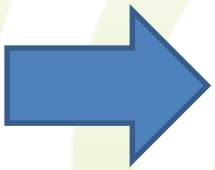
name	salary	pos
Wolf-Tilo	75000	2
Christoph	45000	4

9.4 Ranking Functions

- **RANK()** returns the **rank** of each row in the result list
  - **Example:**

```
SELECT name, salary, RANK() OVER (
   ORDER BY salary DESC
) AS rank
FROM salary
```

person	name	salary
	Christoph	45000
	Wolf-Tilo	75000
	Larry	200000000
	Joachim	45000



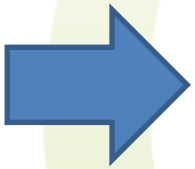
name	salary	rank
Larry	200000000	1
Wolf-Tilo	75000	2
Joachim	45000	3
Christoph	45000	3

9.4 Ranking Functions

- **DENSE_RANK()** works like **RANK()** but does not skip ranks on ties (as it is usually done)
- **Example:**

```
SELECT name, salary, RANK() OVER (
  ORDER BY salary ASC
) AS rank, DENSE_RANK() OVER (
  ORDER BY salary ASC
) AS drank
FROM salary
```

person	name	salary
	Christoph	45000
	Wolf-Tilo	75000
	Larry	200000000
	Joachim	45000



name	salary	rank	drank
Christoph	45000	1	1
Joachim	45000	1	1
Wolf-Tilo	75000	3	2
Larry	200000000	4	3

9.4 CASE Expressions

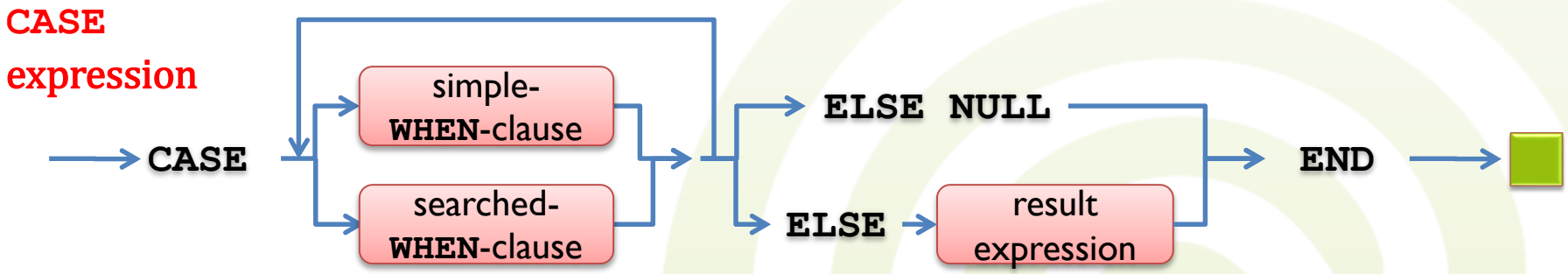
- Very often **codes** are used for storing more complex information
  - Retrieving the account information for owner "Clark" with appropriate account descriptions needs a join
  - Indicate all customers with a negative balance with the string "not creditworthy" in the query result

account	owner	balance	type
	Clark	367,00	0
	Louis	-675,00	0
	Clark	54987,00	1

acc_type	type	description
	0	checking account
	1	savings account
	2	credit card account

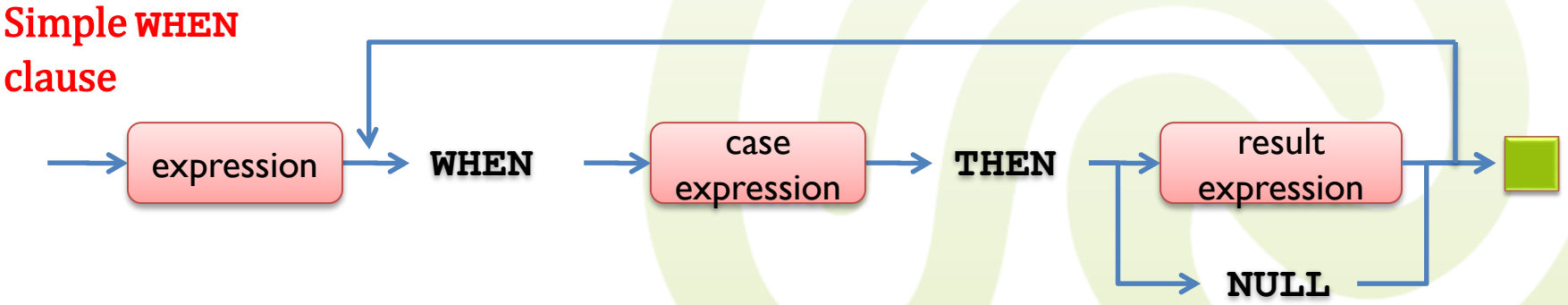
9.4 CASE Expressions

- The **CASE expression** allows a value to be selected based on the evaluation of one or more conditions (similar to "if-then-else")
  - Comes in two flavors:



9.4 CASE Expressions

- The **simple `WHEN` clause:**
  - **Compares an expression to each case expression** one by one
    - If expression is equal to search value, the corresponding result expression is returned
  - If **no match is found,** then some default (**`ELSE`** clause) is returned
    - If **`ELSE`** is omitted, then **`NULL`** is returned



9.4 CASE Expressions

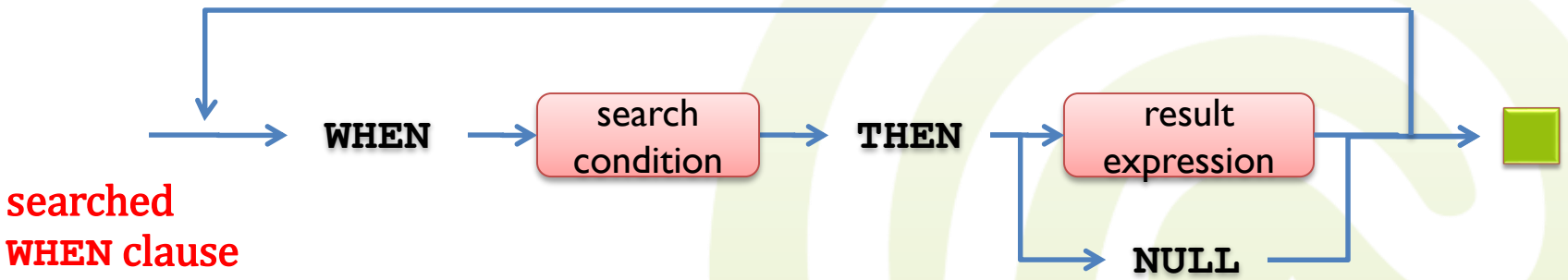
- **Example:** simple **WHEN** clause
  - Directly decode the account type

```
SELECT owner,
   CASE type
      WHEN 0 THEN 'checking account'
      WHEN 1 THEN 'savings account'
      WHEN 2 THEN 'credit card account'
   END
FROM account
```

account	owner	balance	type
	Clark	367,00	0
	Louis	-675,00	0
	Clark	54987,00	1

9.4 CASE Expressions

- The **searched `WHEN` clause**:
  - Checks search conditions from left to right
  - **Stops** as soon as a **search condition evaluates to true**
    - Returns the corresponding result then
  - If **no condition is true**, the value given by the **`ELSE`** clause is returned (or **`NULL`**, if there is no **`ELSE`** clause)



9.4 CASE Expressions

- **Example:** searched **WHEN** clause
  - Retrieve credit rating of customers based on their checking accounts
  -
```
SELECT owner,
   CASE
      WHEN balance < 0 THEN 'not credit-worthy'
      WHEN balance = 0 THEN 'questionable'
      ELSE 'credit-worthy'
   END
FROM account
WHERE type = 0
```

account	owner	balance	type
	Clark	367,00	0
	Louis	-675,00	0
	Clark	54987,00	1

9.5 More on SQL

- There are **many more SQL** statements than we are covering in our lectures
- Moreover, there are many different SQL dialects
- If you don't want to get mad, do the following:
  - Don't care too much about the SQL standard (unless you are actually implementing an RDBMS)
  - Read the **SQL manuals** of **your RDBMS**!

9.5 More on SQL

- **Example:** DB2's SQL reference (version 9.7)
  - http://publib.boulder.ibm.com/infocenter/db2luw/v9r7