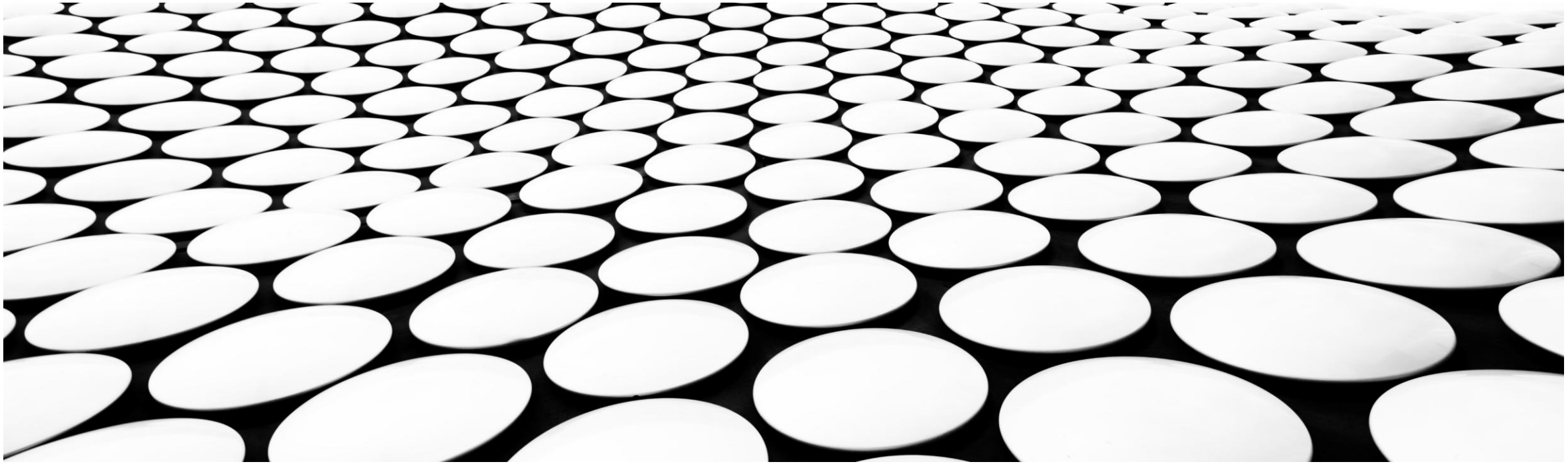
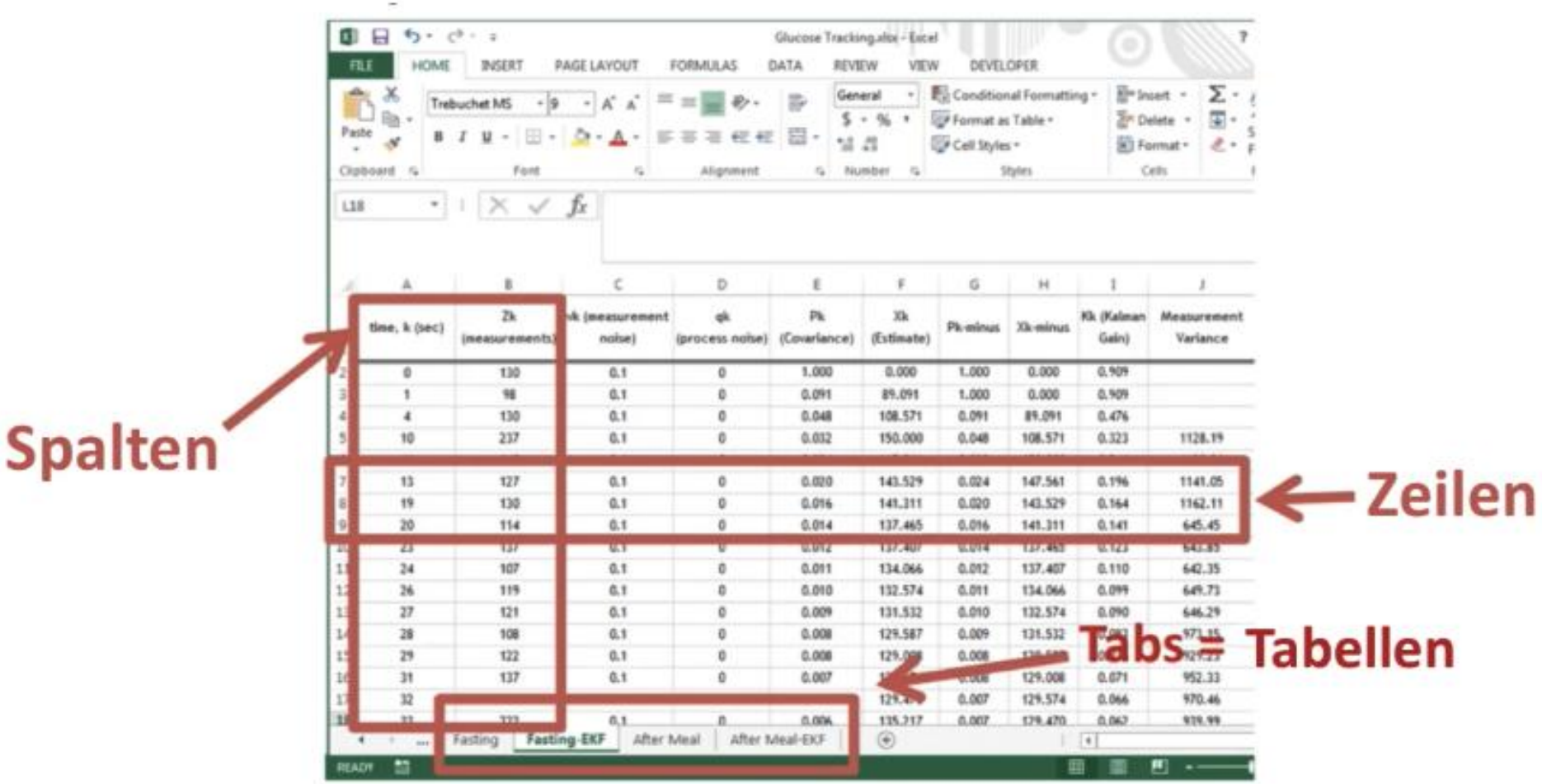
ORACLE

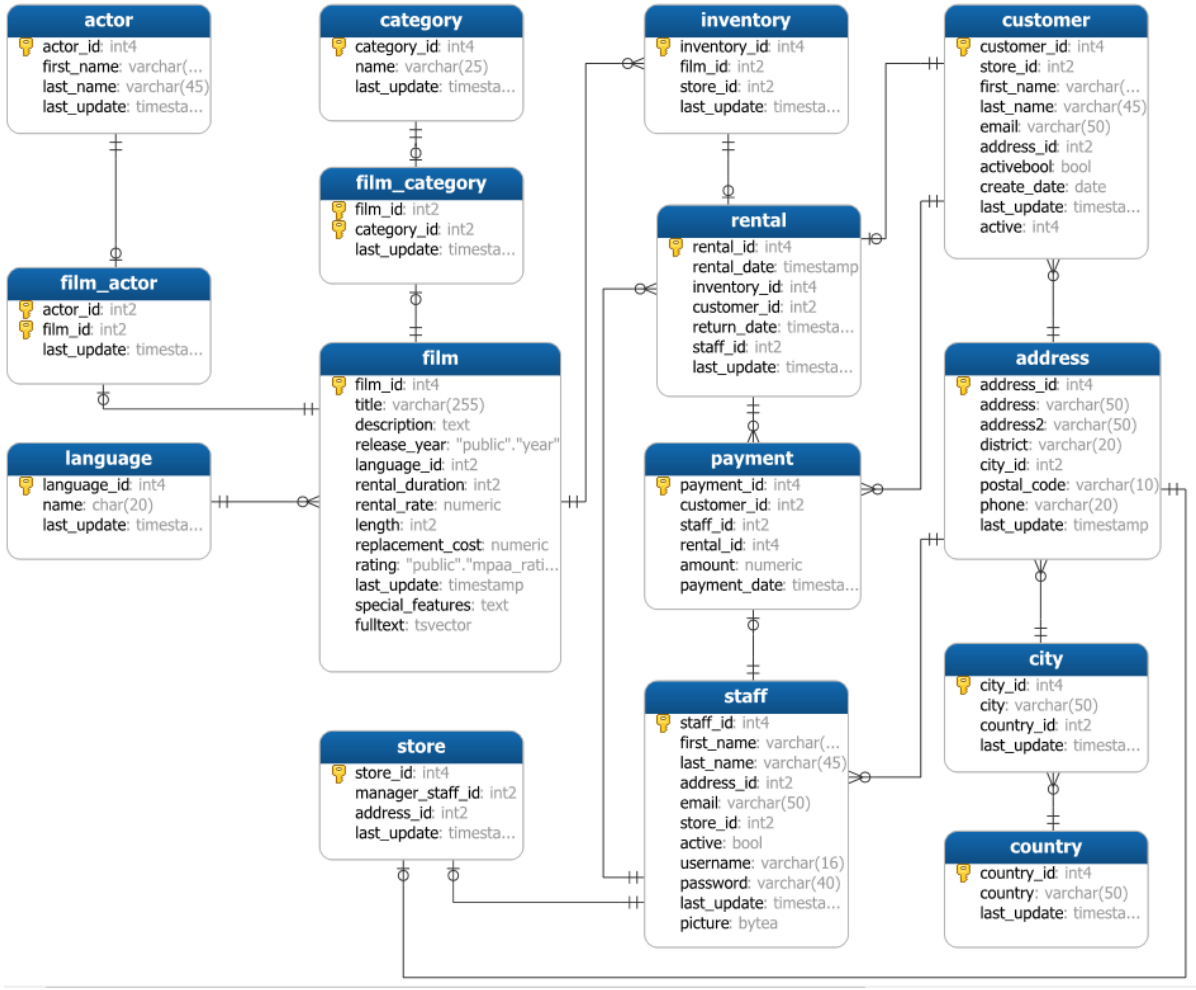
GRUNDLAGEN



WAS IST EINE TABELLE?



TABELLENKALKULATION VS. DBMS



DBM-SYSTEME

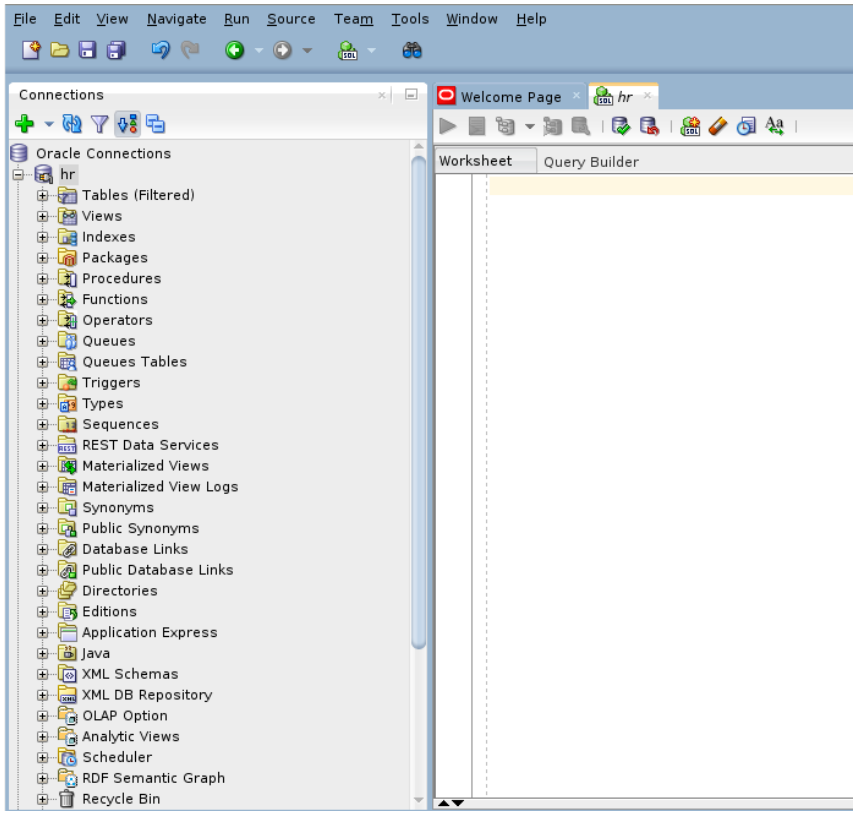
PostgreSQL 	Kostenlos (Open Source) Im Internet häufig benutzt Multiplattform
MySQL MariaSQL  	Kostenlos (Open Source) Im Internet häufig benutzt Multiplattform
MS SQL Server Express 	Kostenlos aber mit einigen Einschränkungen Mit SQL Server kompatibel Nur für Windows
Microsoft Access 	Kostenpflichtig (-) Nicht einfach nur für SQL zu benutzen (-)
SQLite 	Kostenlos (Open Source) Hauptsächlich Kommandozeile (-)

NUTZBARKEIT VON SQL

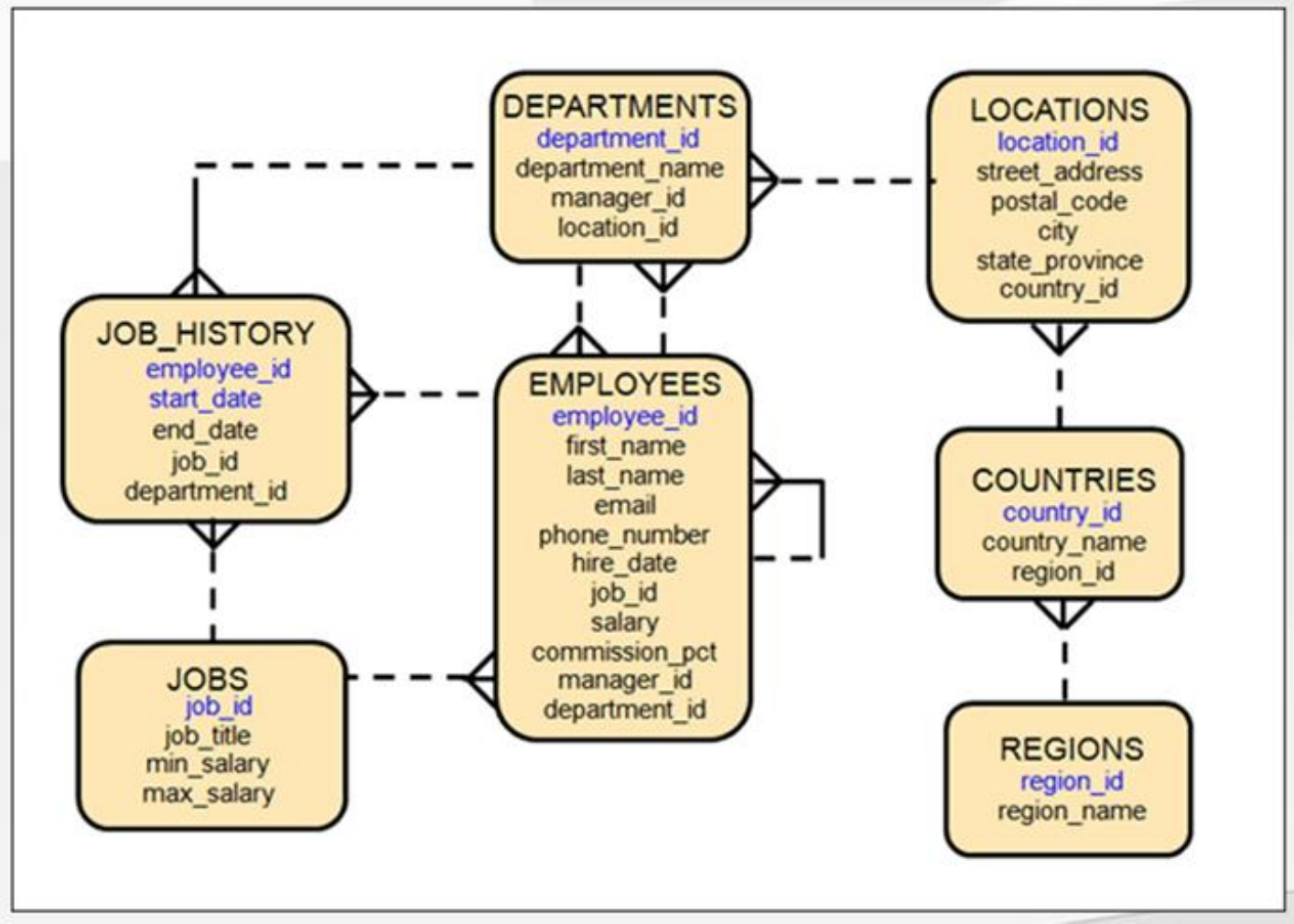
- MySQL
- PostgreSQL
- Oracle Databases
- Microsoft Access
- Looker (BI-Plattform)
- MemSQL (cloud-native database for data-intensive applications)
- Periscope Data (charts)
- Hive (läuft auf Hadoop) (big data)
- Google BigQuery (cloud data warehouse)
- Facebook Presto

WAS IST EIN SCHEMA?

- Ein Schema ist eine Sammlung von Objekten in einer Oracle Datenbank.
- Objektbeispiele:
  - Tabellen
  - Views
  - Triggers
  - Constraints etc.
- Wir arbeiten mit dem Schema „hr“ für Human Resources



TABELLEN IM HR-SCHEMA



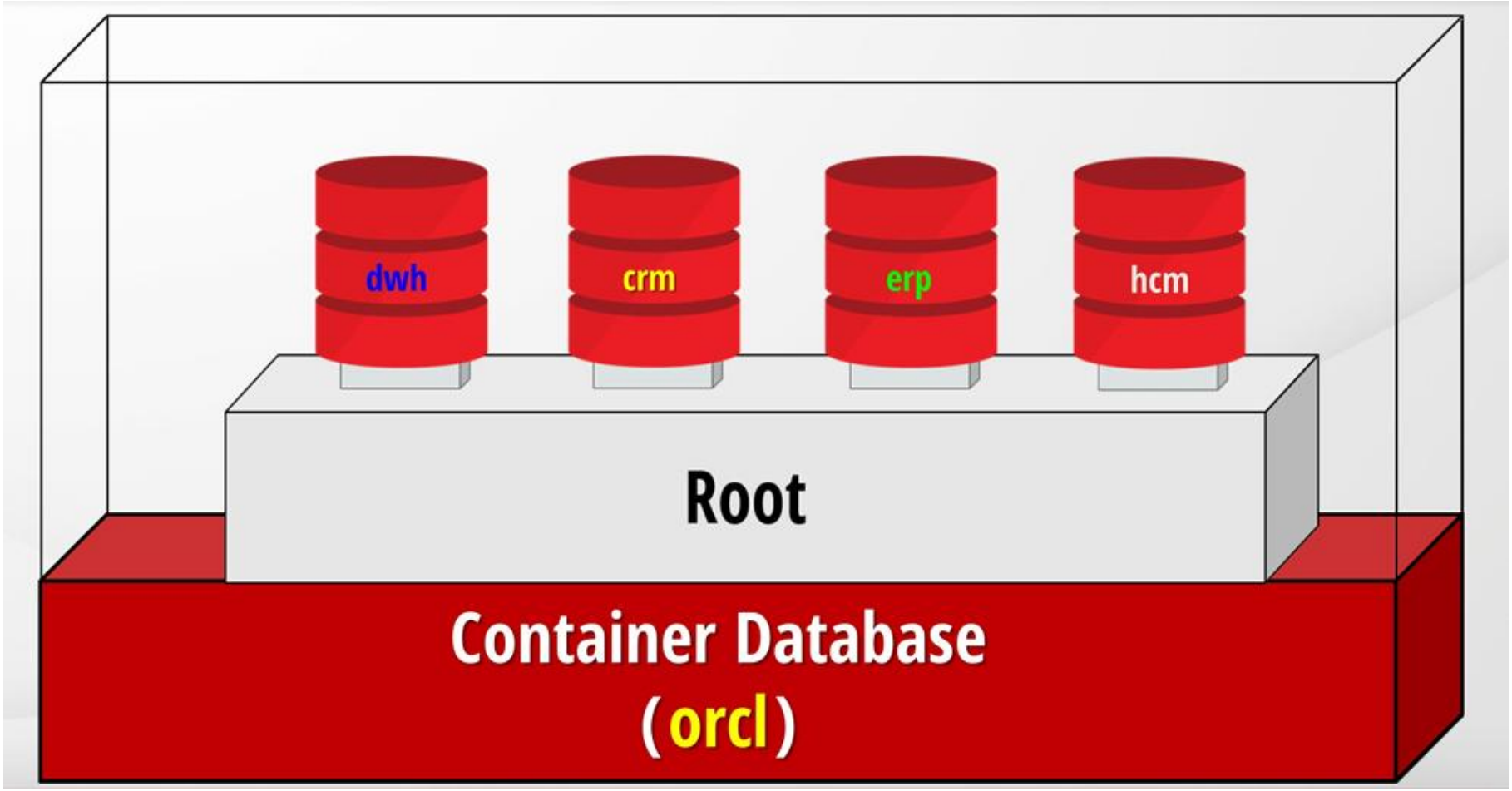
DATABASE OBJECTS

- Oracle bietet Schema objects und Nonschema objects
- Ein Schema ist eine Sammlung von logischen Strukturen von Daten oder Objekten.
  - Tabellen — dient als Basis für das Speichern von Daten in Spalten und Zeilen
  - Views — sind eine „virtuelle Tabelle“, die i.d.R. ausgewählte Spalten und Daten zur Verfügung stellen
  - Constraints — stellen Regeln dar, die bei der Eingabe/Änderung von Daten greifen
  - Indexes — beschleunigen den Zugriff auf Daten; diese werden für ausgewählte Spalten festgelegt
  - Sequences — sind Datenbankobjekte, die einzigartige Integerwerte erzeugen
  - Synonym — ist ein „Alias“ (alternativer Name) für ein Datenbankobjekt
  - Materialized View — ist eine physische Ansicht, die aus einer SQL-Abfrage erstellt wurde und die ursprüngliche Tabelle gelöscht hat.
  - Functions — liefern einen Wert zurück
  - Procedures — liefern nichts zurück
  - Packages — enthalten compilierten Code (PL/SQL), Variablen, Cursors etc., um eine oder mehrere Operationen mit Hilfe von Funktionen oder Procedures auszuführen.

WAS IST SQL

- Structured Query Languages
- Sprache, um mit einer Datenbank zu kommunizieren
- Einsatzgebiete:
  - BI, Data Science, Datenbank Administration, Web Development etc.

PLUGGABLE DATABASE



SQL STATEMENTS

Kategorie	... steht für	... enthält Befehle
DML	Data Manipulation Language	SELECT
		INSERT
		UPDATE
		DELETE
		MERGE
DDL	Data Definition Language	CREATE
		ALTER
		DROP
		RENAME
		TRUNCATE
DCL	Data Control Language	GRANT
		REVOKE
TCL	Transaction Control Lanuage	COMMIT
		ROLLBACK
		SAVEPOINT

BEVOR ES LOSGEHT.

EIN PAAR HINWEISE, DIE HILFREICH SEIN KÖNNEN.

DESCRIBE & INFORMATION

- `DESCRIBE employees;`
  gibt Informationen zur Tabellenstruktur über die angegebene Tabelle an.
- `DESC employees;`
  führt zum gleichen Ergebnis.

- `INFORMATION employees;`
  liefert sehr viel ausführlichere Informationen.
- `INFO employees;`
  führt zum gleichen Ergebnis.

DUAL

- Anders als in anderen DBMS führt der Befehl

  ```
  SELECT ‘Hallo Welt‘;
  ```

  zu einer Fehlermeldung – FROM … wird verlangt.
- Für Ausgaben, die nicht Daten aus einer Tabelle liefern sollen, steht die Tabelle DUAL zur Verfügung.
- DUAL enthält nur eine Spalte und einen Datensatz mit Wert „X“ – ist eine Dummy-Tabelle.

  ```
  SELECT ‘Hallo Welt‘ FROM dual;
  ```

  gibt das gewünschte Ergebnis zurück.
- Entsprechendes gilt für Rechenoperationen, die unabhängig von Tabellen durchgeführt werden sollen.
- Anweisungen IMMER mit „;“ abschließen.
- Ausführen mit `F9` oder `STRG + ENTER`.

SELECT

DIE AUSGABE VON DATEN

SELECT

- … liefert Daten aus einer Datenbank.
- Grundaufbau:

```
SELECT *|{column_name1, column_name2, …} FROM table;
```

- * liefert alle Daten einer Tabelle – ohne Kenntnis der Metadaten

```
SELECT * FROM employees;
```

- Mit Benennung der Spaltennamen werden nur diese ausgegeben.

```
SELECT first_name, last_name FROM employees;
```

SPALTEN ALIAS

- … benennt den Spaltenkopf (column heading) um
- … kann mit oder ohne Keyword AS benutzt werden – AS erhöht Lesbarkeit der Anweisung
- … besonders nützlich bei Kalkulationen
- … besteht i.d.R. aus einem Wort
  Werden mehrere Worte (mit Leerzeichen), Sonderzeichen oder casesensitive Begriffe benutzt, sind Anführungszeichen notwendig.

```
SELECT first_name AS vorname, last_name AS nachname
    FROM employees;

SELECT first_name “MA Vorname“, last_name “MA Nachname“
    FROM emmployees;
```

QUOTE (q) OPERATOR

- Hochkomma verbessert Lesbarkeit und Usability.

```
SELECT q‘[Mein Name ist Jeff und der meines Freundes ist
    Karl]‘  mein_text FROM dual;
```

- Es können alle Zeichen als Begrenzer benutzt werden.

  [ ], { }, ( ), < >, ‘A‘, ‘*‘ …

```
SELECT q‘* Meine Name ist Jeff *‘ text1, Q ‘bMein Name ist
    textb‘ text2 FROM dual;
```

- Bevorzugter Begrenzer: [ ]

DISTINCT & UNIQUE OPERATOR

01_Select_3_Distinct

- ... eliminieren doppelte Werte in Ergebnisliste.

```
SELECT job_id FROM employees;

SELECT DISTINCT job_id FROM employees;

SELECT UNIQUE job_id FROM employees;
```

- Es darf nur ein (!) DISTINCT / UNIQUE in einer Abfrage benutzt werden.

```
SELECT job_id, department_id FROM employees;

SELECT DISTINCT job_id, DISTINCT department_id FROM employees;

SELECT job_id, DISTINCT department_id FROM employees;

SELECT DISTINCT job_id, department_id FROM employees;
```

SPALTEN ALIAS

01_Select_4_Alias

- … benennt den Spaltenkopf (column heading) um
- … kann mit oder ohne Keyword AS benutzt werden – AS erhöht Lesbarkeit der Anweisung
- … besonders nützlich bei Kalkulationen
- … besteht i.d.R. aus einem Wort
  Werden mehrere Worte (mit Leerzeichen), Sonderzeichen oder casesensitive Begriffe benutzt, sind Anführungszeichen notwendig.

```
SELECT first_name AS vorname, last_name AS nachname
    FROM employees;

SELECT first_name “MA Vorname“, last_name “MA Nachname“
    FROM emmployees;
```

CONCATINATION OPERATOR

01_Select_5_Concat

- ... verbinden zwei oder mehr Spaltenwerte und gibt diese in einer Spalte zurück.
- || verbindet zwei oder mehrere Spaltenwerte

```
SELECT first_name || ' ' || last_name FROM employees;
```

- Eine Verbindung mit einem NULL-Wert liefert keinen NULL-Wert zurück, sondern den vorhandenen Wert.

```
SELECT first_name || ' ' || manager_id FROM employees;
```

- Die Verwendung von ALIAS erhöht Lesbarkeit

```
SELECT first_name || ' ' || manager_id
    AS “Vorname und Manager-ID” FROM employees;
```

ARITHMETIC EXPRESSIONS

- ... werden eingesetzt, um arithmetische Operationen in SQL durchzuführen.
- ... kann Spaltennamen, Zahlen und arithmetische Operatoren enthalten.

Operator	Beschreibung
+	Addition
-	Subtraktion
*	Multiplikation
/	Division

➔ Multiplikation & Division werden VOR Addition & Substraktion ausgeführt.

➔ Klammern haben die höchste Priorität.

➔ Klammern erhöhen die Lesbarkeit!

```
SELECT employee_id, salary, salary+50*12
    AS “Jahreseinkommen“ FROM employees;
```

- Arithmetische Operationen mit Datumswerten liefern neuen Datumswert zurück.
- Arithmetische Operationen mit NULL-Werten liefern NULL-Werte zurück. Lösung: nvl(Formel,0)

FILTERN & SORTIEREN VON DATEN

WHERE KLAUSEL

- … schränkt die Ausgabe der Zeilen ein.
- … wird benutzt mit:
  - Vergleichsoperatoren (=, <, >, <=, >=, <>, !=)
  - BETWEEN … AND, IN, LIKE und NULL
  - Logischen Operatoren (AND, OR, NOT)

```
SELECT * FROM employees;
```
➔ 107 Zeilen / Datensätze

```
SELECT * FROM employees WHERE job_id = 'IT_PROG'
```
➔ 5 Zeilen / Datensätze

- BETWEEN … AND
  - … liefert Daten, die zwischen dem niedrigsten und höchsten Begrenzungswert liegen
  - Begrenzungswerte sind inkludiert! (ACHTUNG: gilt nicht für Zeichenkriterien! Hier ist AND nicht mit inbegriffen.)
  - Zahlen, Datumswerte, Zeichenwerte können hiermit gefiltert werden.

WHERE

02_Where_1

- In der WHERE Klausel dürfen keine Aggregatfunktionen benutzt werden.
- WHERE Klausel steht VOR der GROUP BY und ORDER BY Klausel.
- Vergleichswerte sind Case-sensitive

IN OPERATOR

02_Where_2_In

- ... liefert Daten zurück, die einer Liste entsprechen.
- ... kann auf Zahlen, Datumswerte, Zeichenfolgen angewendet werden.
- Die Reihenfolge der Werte in der Liste ist NICHT entscheidend.

```
SELECT * FROM employees WHERE employee_id IN (100, 105, 102, 200)
```

LIKE OPERATOR

- … sucht nach einer Teilzeichenkette in Zeichenketten.
- … ermöglicht Suche mit Wildcards.
- ACHTUNG: Begrenzer mit (einfachen) Hochkommata!
- Wildcard-Operatoren von Oracle:

Symbol	Beschreibung
%	Berücksichtigt die Datensätze, auf die die Suche mit keinem oder mehreren Zeichen zutrifft.
_	Listet Datensätze, auf die genau ein Zeichen zutrifft

- LIKE kann auch *ohne* Wildcard eingesetzt werden – ist aber sinnlos, weil dies „=“ entspricht.
- LIKE ist langsamer als „=“.

```
SELECT first_name FROM employees WHERE first_name LIKE ‘A%‘
SELECT first_name FROM employees WHERE job_id LIKE ‘SA_%‘
```

IS NULL OPERATOR

- ... sucht nach NULL-Werten.
- = NULL ist nicht das Gleiche wie IS NULL

```
SELECT first_name, last_name, manager_id
    FROM employees
        WHERE manager_id = NULL;
```

```
SELECT first_name, last_name, manager_id
    FROM employees
        WHERE manager_id IS NULL;
```

- IS NOT NULL schließt NULL-Werte aus.

```
SELECT first_name, last_name, manager_id
    FROM employees
        WHERE manager_id IS NOT NULL;
```

LOGISCHE OPERATOREN . AND

02_Where_5_And

- ... ermöglichen die Anwendung von mehreren Bedingungen in der WHERE-Klausel.

Operator	Bedeutung
AND	liefert TRUE zurück, wenn alle Bedingungen erfüllt sind.
OR	liefert TRUE zurück, wenn mindestens eine Bedingung erfüllt ist.
NOT	liefert TRUE zurück, wenn Bedingung nicht erfüllt ist.

- AND Operator – Kombinationsmöglichkeiten

AND	TRUE	FALSE	NULL
TRUE	TRUE	FALSE	NULL
FALSE	FALSE	FALSE	FALSE
NULL	NULL	FALSE	NULL

```
SELECT first_name, job_id, salary FROM employees WHERE job_id = ,‘IT_PROG‘  AND salary >= 5000;
```

LOGISCHE OPERATOREN . OR

- Beim OR Operator muss mindestens eine Bedingung erfüllt sein.
- Kombinationsmöglichkeiten:

OR	TRUE	FALSE	NULL
TRUE	TRUE	TRUE	TRUE
FALSE	TRUE	FALSE	NULL
NULL	TRUE	NULL	NULL

```
SELECT first_name, job_id, salary
    FROM employees
        WHERE job_id = ‘IT_PROG‘ OR salary >= 5000;
```

LOGISCHE OPERATOREN . NOT

02_Where_7_Not

- NOT Operator wird eingesetzt, um Bedingung zu negieren.
- Kombinationsmöglichkeiten:

NOT	TRUE	FALSE	NULL
	FALSE	TRUE	NULL

```
SELECT last_name, job_id, salary
    FROM employees
       WHERE salary > 10000
          AND job_id NOT IN ('SA_MAN', 'ST_CLERK', 'SH_CLERK')
```

REIHENFOLGE DER ABARBEITUNG (WHERE)

Reihenfolge	Operatoren
1	Arithmetic Operators
2	Concatination Operator
3	Comparison Conditions
4	IS [NOT] NULL, LIKE, [NOT] IN
5	[NOT] BETWEEN
6	Not Equal To
7	NOT
8	AND
9	OR

Klammern können die Reihenfolge der Berücksichtigung beeinflussen, da diese prioritär ausgeführt werden.

Vergleiche:

```
SELECT last_name, job_id, salary FROM employees WHERE job_id ='IT_PROG' OR job_id='ST_CLERK' AND salary > 5000;
SELECT last_name, job_id, salary FROM employees WHERE (job_id ='IT_PROG' OR job_id='ST_CLERK') AND salary > 5000;
```

ORDER BY

- … beeinflusst die Sortierreihenfolge zurückgegebener Zeilen durch Angabe von
  - Spaltennamen
  - Alias oder
  - Positionsangabe
- Es kann aufsteigend (ASC; Standardeinstellung) oder absteigend (DESC) für jede Spalte sortiert werden.
- Kann mit dem SELECT-Statement genutzt werden.
- … ändert nicht die Reihenfolge der Daten in der Tabelle, sondern nur in den zurückgegebenen und angezeigten Zeilen.
- Steht immer am Ende der SELECT-Anweisung.
- NULL-Werte stehen immer am Ende einer aufsteigend sortierten Ausgabe.
- Mit `NULLS FIRST` and `NULLS LAST` Operatoren kann Platzierung der NULL-Werte beeinflusst werden.
- Grundaufbau:

```
SELECT col1, col2, … FROM tbl WHERE condition ORDER BY col1 [, 2, ALIAS, ];
```

ROWID UND ROWNUM

ROWID	ROWNUM
• Die ROWID ist ein eindeutiger Kennzeichner, der die physikalische Adresse einer Zeile enthält. • Wird automatisch durch Oracle beim Einfügen einer neuen Zeile geniert. • Stellt den schnellsten Weg dar, um eine einzelne Zeile anzusprechen. • Die ROWID ist permanent. • Sie ändert sich nicht.	• DIE ROWNUM ist eine logische sequentielle Nummer für eine Zeile, die bei Rückgabe einer Anfrage vergeben wird. • Um die Anzahl der zurückzugebenden Zeilen zu limitieren, kann diese Pseudospalte genutzt werden. • ROWNUM ist temporär. • Sie ändert sich mit Änderung der Abfrage.

FETCH

- … wird in Verbindung mit SELECT und ORDER BY genutzt, um die Anzahl der auszugebenden Zeilen zu beeinflussen.
- Grundaufbau:

```
SELECT …
  ORDER BY …
      [OFFSET 10 rows]
          FETCH [FIRST | NEXT] [Anzahl Zeilen | Prozentzahl PERCENT] ROW(s)] [ONLY | WITH TIES];
```

- `OFFSET 10 rows` überspringt ersten 10 Zeilen
- `FETCH FIRST 10 ROWS ONLY` zeigt Zeilen 1-10 an
- `FETCH FIRST 10 PERCENT ROWS ONLY` zeigt die ersten 10 Prozent der Datensätze an

SUBSTITUTION VARIABLES

- … stellen abgefragte User Variablen dar.
- … sind ein Platzhalter für einen Variablenwert, der vom User abgefragt wird.
- & wird vor dem Variablen genutzt

```
SELECT employee_id, first_name, last_name FROM employees
        WHERE employee_id = &emp_no;
```

- Wenn ein String oder ein Datum abgefragt wird, muss die Variable in Hochkomma stehen.

```
SELECT first_name, last_name FROM employees
        WHERE first_name = '&vorname'
```

- Es können mehrere Substitution Variables an ganz verschiedenen Stellen in einer Abfrage genutzt werden.

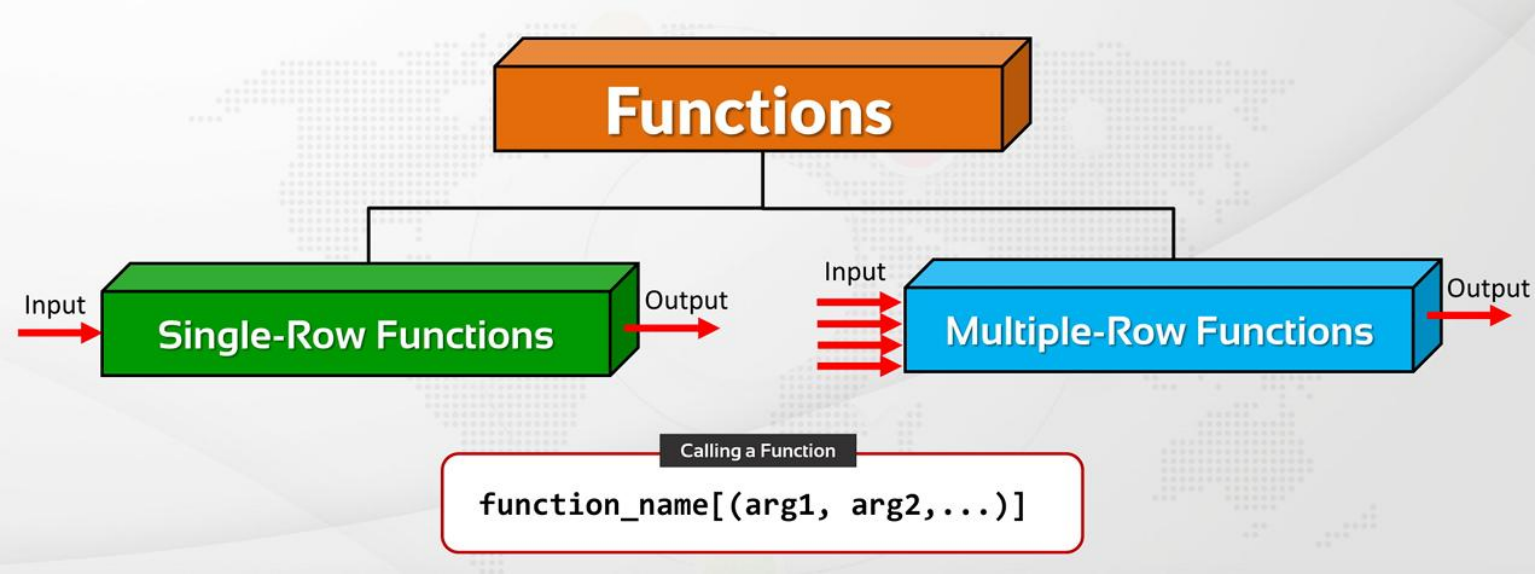
```
SELECT employee_id, first_name, last_name, &col_name
        FROM &table_name
        WHERE &condition
        ORDER BY &order_clause;
```

SINGLE ROW FUNCTIONS

NICHTS FÜR PAARE

WAS SIND FUNKTIONEN?

- Funktionen werden zur Manipulation von Daten und zur Rückgabe von Werten genutzt.
- … müssen vor ihrem Aufruf erstellt worden sein.
- Der Aufruf erfolgt über die Angabe des Funktionsnamens & ggf. weiteren Parametern.
- … sind zur Wiederverwendung erstellt worden.
- Typen von Funktionen:
  - Single-Row Functions
  - Multiple-Row Functions



SINGLE-ROW FUNCTIONS

- … agieren auf einer Zeile und geben auch nur einen Wert zurück.
- … akzeptieren einen oder mehrere Parameter, um einen Wert zurückzugeben.
- … geben für jede Zeilen einen Wert zurück.
- … können allein oder eingebettet („nested“) genutzt werden.
- Rückgabewert kann sich vom Datentyp her von Eingabewert unterscheiden.
- … kann genutzt werden in SELECT, WHERE oder ORDER BY – Klausel.
- … werden kategorisiert nach Datentyp des Eingabewertes:
  - Character Functions — Eingabewert: Zeichen – Ausgabewert: Zeichen oder Zahlen
  - Number Functions — Eingabewert: Zahl – Ausgabewert: Zahl
  - Date Functions — Arbeiten mit Datumswerten
  - Conversion Functions — Verwandeln einen Datentyp in einen anderen
  - Generell Functions — Können jeden Datentyp behandeln – werden aber i.d.R. zur Handhabung von NULL-Werten genutzt.

CHARACTER FUNCTIONS

- … haben Zeichen als Eingabewert und Zeichen oder Zahlen als Ausgabewert.
- Unterscheidung in 2 Kategorien:

1. Case Conversion functions:
    1. UPPER() Umwandlung Zeichen in Großbuchstaben
    2. LOWER() Umwandlung Zeichen in Kleinbuchstaben
    3. INITCAP() Umwandlung Zeichen in ersten großen Buchstaben
2. Character Manipulation functions:
    1. SUBSTR() Auswahl von Teilzeichenkette aus einem String
    2. LENGTH() ermittelt Länge von Zeichen
    3. CONCAT() verbindet 2 oder mehr Zeichenketten miteinander
    4. INSTR() ermittelt Zeichen in einer Zeichenkette
    5. TRIM() löschen Leerzeichen vor und nach Zeichenkette
    6. REPLACE() ersetzen angegebene Zeichen mit neuen Zeichen

CHARACTER FUNCTIONS

Character Manipulation Functions Syntax	Example	Result
SUBSTR(source_string, position[,length])	SUBSTR('Sql Course',1,3)	Sql
LENGTH(string)	LENGTH('Sql Course')	10
CONCAT(string1,string2)	CONCAT('Sql','Course')	SqlCourse
INSTR(string,substring[,position,occurrence])	INSTR('Sql Course','o')	6
TRIM([[LEADING|TRAILING|BOTH] trim_character FROM] string)	TRIM(' Sql Course ')	Sql Course
LTRIM(string,[trim_string])	LTRIM(' Sql Course ')	Sql Course
RTRIM(string,[trim_string])	RTRIM(' Sql Course ')	Sql Course
REPLACE(string,string_to_replace[,replacement_string])	REPLACE('Sql Course','s','*')	Sql Cour*e
LPAD(string,target_length,padding_expression)	LPAD('sql',10,'-')	-------sql
RPAD(string,target_length,padding_expression)	RPAD('sql',10,'-')	sql-------

DATE FUNCTIONS

Date Functions	Meanings
ADD_MONTHS(date, n)	Adds months to a date.
MONTHS_BETWEEN(date1,date2)	Number of months between 2 dates.
ROUND(date[,format])	Rounds a date/time value to a specified element.
TRUNC(date[,format])	Truncates a date/time value to a specific element.
EXTRACT(date_component FROM date)	Extracts a specific time component from a date.
NEXT_DAY(date,day_of_week)	Returns the date of the next specified weekday.
LAST_DAY(date)	Returns the last day of the month.

Examples	Result
ADD_MONTHS('18-SEP-23', 3)	18-DEC-23
MONTHS_BETWEEN('03-SEP-20','18-FEB-20')	6.516129032258064516129032258064516129032
ROUND(sysdate,'MONTH')	01-JUL-20
TRUNC(sysdate,'YEAR')	01-JAN-20
EXTRACT(month FROM sysdate)	6
NEXT_DAY('04-JUN-20','TUESDAY')	09-JUN-20
LAST_DAY('04-JUL-20')	31-JUL-20

CONDITIONAL EXPRESSIONS

CASE … WHEN EXPRESSIONS

- … liefert eine IF-THEN-ELSE- Logik in einem SQL-Statement

```
CASE expression WHEN comparison_expression_1 THEN result_1
                [WHEN comparison_expression_1 THEN result_1
                …
                WHEN comparison_expression_1 THEN result_n]
                ELSE result]
END
```

- Expression und comparision_expression müssen vom gleichen Datentyp sein.
- … können in SELECT und WHERE genutzt werden.

CASE EXPRESSION IM SELECT

```
SELECT first_name,last_name, job_id, salary,hire_date,
    CASE job_id WHEN 'ST_MAN' THEN 1.20 * salary
                WHEN 'SH_MAN' THEN 1.30 * salary
                WHEN 'SA_MAN' THEN 1.40 * salary
                ELSE salary END "UPDATED_SALARY"
FROM EMPLOYEES WHERE job_id
IN ('ST_MAN','SH_MAN','SA_MAN');
```

	FIRST_NAME	LAST_NAME	JOB_ID	SALARY	HIRE_DATE	UPDATED_SALARY
1	John	Russell	SA_MAN	14000	01-OCT-04	19600
2	Karen	Partners	SA_MAN	13500	05-JAN-05	18900
3	Alberto	Errazuriz	SA_MAN	12000	10-MAR-05	16800
4	Gerald	Cambrault	SA_MAN	11000	15-OCT-07	15400
5	Eleni	Zlotkey	SA_MAN	10500	29-JAN-08	14700
6	Matthew	Weiss	ST_MAN	8000	18-JUL-04	9600
7	Adam	Fripp	ST_MAN	8200	10-APR-05	9840
8	Payam	Kaufling	ST_MAN	7900	01-MAY-03	9480
9	Shanta	Vollman	ST_MAN	6500	10-OCT-05	7800
10	Kevin	Mourgos	ST_MAN	5800	16-NOV-07	6960

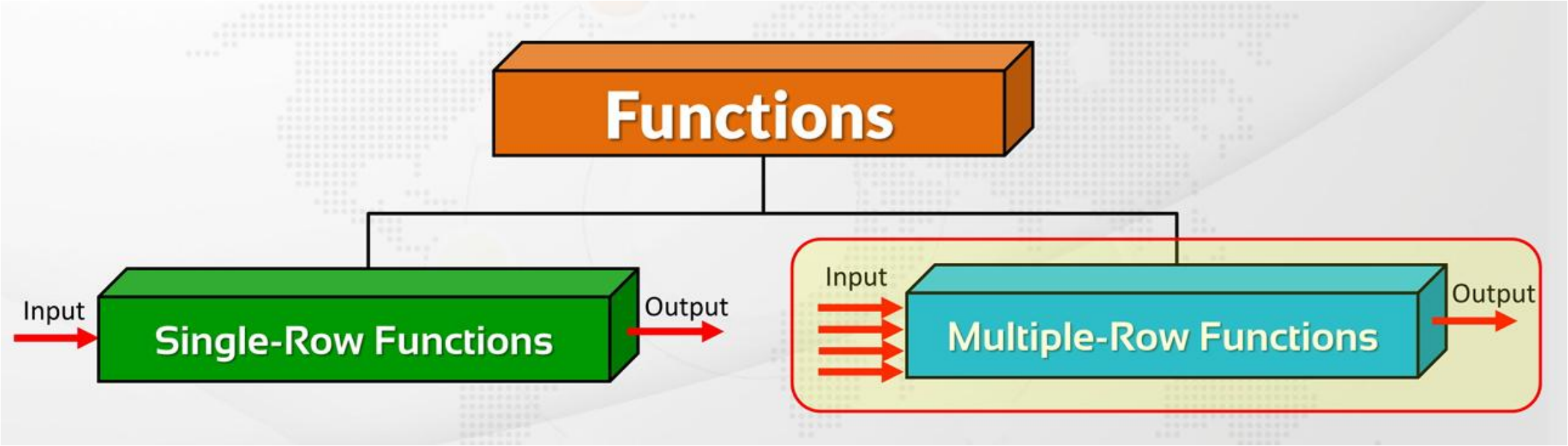
CASE EXPRESSION IN WHERE

```
SELECT first_name, last_name, job_id, salary
FROM employees
WHERE
  (CASE
      WHEN job_id = 'IT_PROG' AND salary > 5000  THEN 1
      WHEN job_id = 'SA_MAN'  AND salary > 10000 THEN 1
    ELSE 0
  END) = 1 ;
```

	FIRST_NAME	LAST_NAME	EMAIL	JOB_ID	SALARY
1	Alexander	Hunold	AHUNOLD	IT_PROG	9000
2	Bruce	Ernst	BERNST	IT_PROG	6000
3	John	Russell	JRUSSEL	SA_MAN	14000
4	Karen	Partners	KPARTNER	SA_MAN	13500
5	Alberto	Errazuriz	AERRAZUR	SA_MAN	12000
6	Gerald	Cambrault	GCAMBRAU	SA_MAN	11000
7	Eleni	Zlotkey	EZLOTKEY	SA_MAN	10500

AGGREGAT FUNKTIONEN

FUNKTIONSTYPEN



GRUPPIERUNGS FUNKTIONEN

- AVG gibt Durchschnittswert zurück
- COUNT gibt Anzahl Zeilen aus einer Abfrage zurück
- MAX gibt größten Wert aus einer Expression oder Spalte zurück
- MIN gibt kleinsten Wert aus einer Expression oder Spalte zurück
- SUM gibt Summe aus einer Expression oder Spalte zurück
- LISTAGG transformiert und ordnet Daten aus vielen Zeilen in eine Liste von Werten, separiert durch Delimiter

```
SELECT LISTAGG(first_name,‘, ‘) WITHIN GROUP(ORDER BY first_name) “Employees“
FROM employees WHERE job_id = ‘IT_PROG‘;
```

GRUPPIERUNG VON DATEN

GROUP BY KLAUSEL

GROUP BY

- Zur Gruppierung von Zeilen kann GROUP BY benutzt werden.
- Es kann mehr als eine Spalte gruppiert werden.
- Die SELECT Klausel kann keine weiteren Spalten haben als die, die in der GROUP BY Klausel stehen.
- Group functions (z.B. MAX(...)) müssen nicht gruppiert werden.
- Es können so viele Group functions genutzt werden, wie wir wollen.
- Aliasse können nicht in der GROUP BY Klausel genutzt werden.
- Die ORDER BY Klausel kann keine weiteren Spalten enthalten als die, die in der GROUP BY Klausel stehen.
- Mit der WHERE Klausel können die zurückgegebenen Zeilen eingeschränkt werden.

HAVING

- Group functions können nicht in der WHERE Klausel benutzt werden.
- Mit HAVING können Rückgabezeilen gefiltert werden, nachdem diese gruppiert worden sind.
- WHERE filtert Zeilen, HAVING **filtert gruppierte Daten**.
- In der HAVING Klausel können Group functions genutzt werden

```
SELECT job_id, AVG(salary)
FROM employees
GROUP BY job_id
HAVING AVG(salary) > 5000;
```

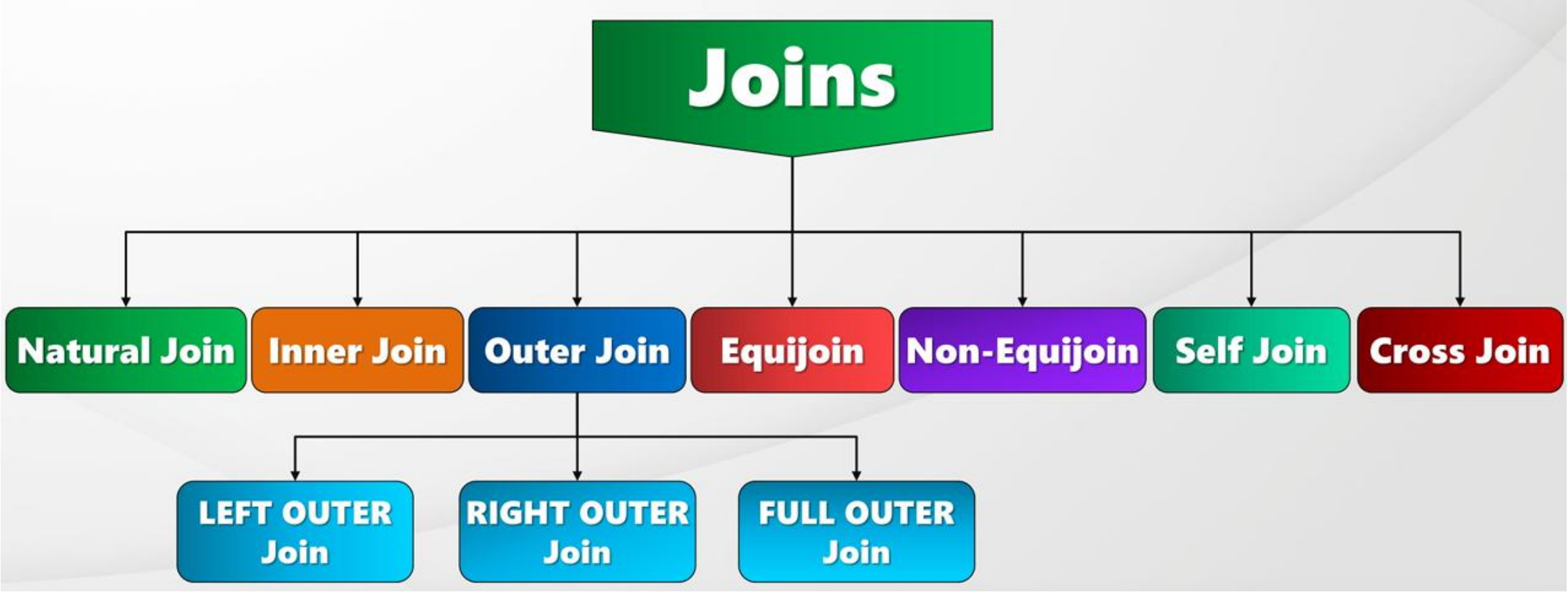
NESTED GROUP FUNCTIONS

- Group functions können eingebettet sein in andere (nested).
- Der Output der eingebetteten Group function ist Input für die äußere Group function.
- Bei Nested Group functions muss GROUP BY benutzt werden.
- Es können maximal 2 Group functions eingebettet sein.

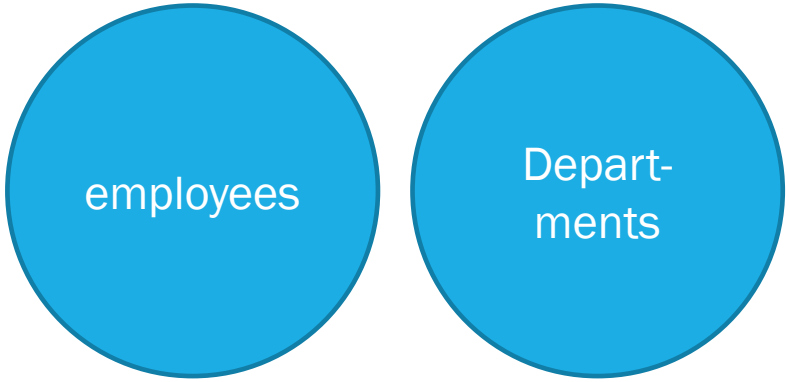
```
SELECT MAX(AVG(salary))
FROM employees
GROUP BY department_id;
```

MEHRERE TABELLEN „JOINEN“

JOIN TYPEN

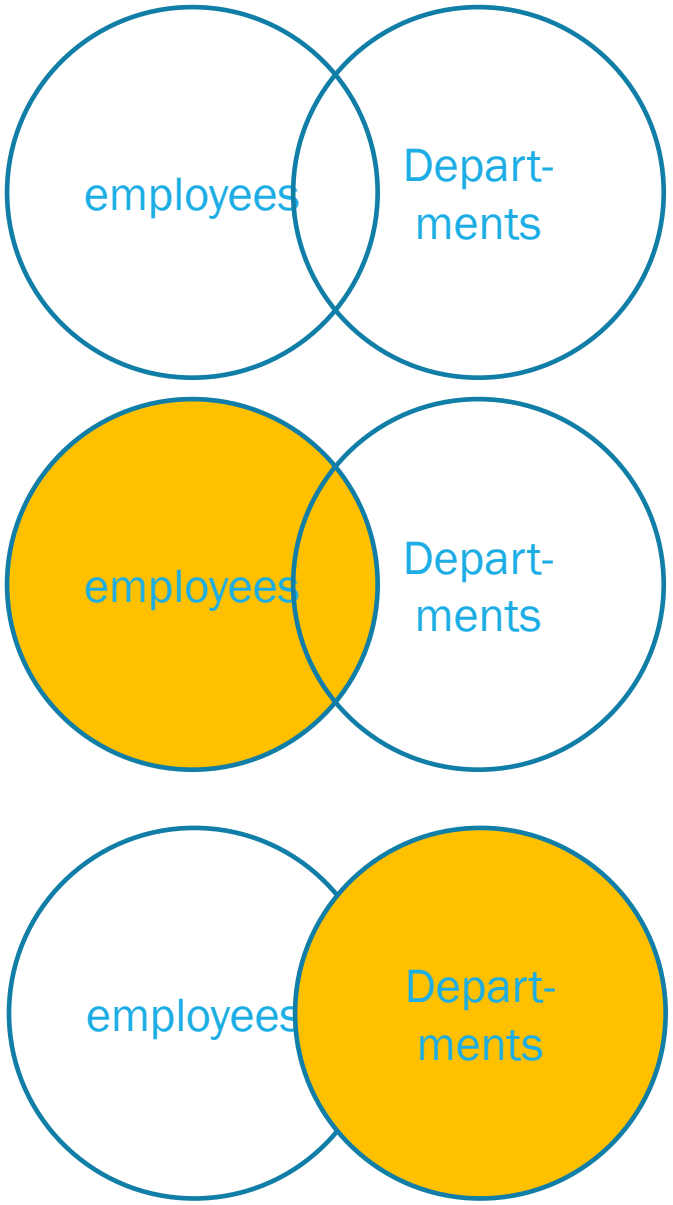


JOIN TYPEN



Department_id
Manager_id

1. Nur überstimmende Wert
2. Alle employees-Werte
3. Alle department-Werte
4. Alle employees- & alle department-Werte



NATURAL JOIN

- … verbindet Tabellen über Spalten, die in beiden Tabellen den gleichen Spaltennamen und –typ haben.
- … verbindet 2 Spalten, die den gleichen Wert haben.
- WHERE Klausel kann zurückzugebende Zeilen einschränken.
- Grundaufbau:

```
SELECT colums FROM table NATURAL JOIN table2;

SELECT * FROM employees NATURAL JOIN departments;
```

- Spalten mit gleichem Namen werden nur einmal ausgegeben.

JOINEN MIT DER USING KLAUSEL

- Wenn es mehr als eine Spalte in zwei Tabellen gibt, die den gleichen Namen haben, kann mit USING eine Spalte ausgewählt werden, über die die Verbindung stattfinden soll.
- Die USING Klausel zählt zu den „Equijoins“.

```
SELECT first_name, last_name, department_name, department_id
FROM employees JOIN departments USING (department_id)
```

INNER JOIN

- ... liefert Zeilen aus beiden Tabellen zurück, die die Join-Konditionen erfüllen oder den Ausdrücken entsprechen (ON / USING).

```
SELECT e.first_name, e.last_name, d.manager_id, d.department_name
FROM employees e JOIN departments d
ON (e.department_id = d.department_id AND e.manager_id = d.manager_id);
```

- Die ON Klausel funktioniert auch, wenn die verbundenen Spalten unterschiedliche Datentypen haben.

MEHRERE JOINS NUTZEN

- Es können mehr als zwei Tabellen verbunden werden.
- Es kann mit USING, ON oder NATURAL JOIN gearbeitet werden.

```
SELECT e.first_name, e.last_name, d.department_name, l.city, l.street_adress, country_id
FROM employees
JOIN departments d
ON (e.department_id = d.department_id)
JOIN locations l
USING (location_id)
NATURAL JOIN countries;
```

JOINS EINSCHRÄNKEN

- ... können Sie mit der WHERE Klausel oder dem AND Operator.

```
SELECT e.first_name, e.last_name, d.department_name, l.city, l.street_adress, country_id
FROM employees
JOIN departments d
ON (e.department_id = d.department_id)
JOIN locations l
ON (d.location_id = l.location_id)
WHERE d.department_id = 100;

SELECT e.first_name, e.last_name, d.department_name, l.city, l.street_adress, country_id
FROM employees
JOIN departments d
ON (e.department_id = d.department_id)
JOIN locations l
ON (d.location_id = l.location_id)
AND d.department_id = 100;
```