Spaß mit Datenbanken

Erstellen der Tabellenstruktur

- Erstellen mit `CREATE TABLE`

```
CREATE TABLE IF NOT EXISTS countries(
    country_id CHAR(2) NOT NULL,
    country_name VARCHAR(40),
    region_id INT,
    PRIMARY KEY(country_id)
    FOREIGN KEY(region_id) REFERENCES regions(region_id));
```

Ändern der Tabellenstruktur

- Nachträgliche Änderungen mit `ALTER TABLE`

BSP:

```
ALTER TABLE locations ADD (inhabitants INT(5));
ALTER TABLE locations MODIFY inhabitants INT(12);
ALTER TABLE locations DROP COLUMN inhabitants;
ALTER TABLE employees ADD FOREIGN KEY(manager_id) REFERENCES
employees(employee_id));
```

Ändern der Tabellenstruktur

- Nachträgliche Löschung mit `DROP TABLE`

  BSP:

```
DROP TABLE locations;
DROP TABLE locations CASCADE;
```

- Mit CASCADE werden ‘abhängige’ Objekte ebenfalls gelöscht

Hinzufügen von Daten

```
INSERT
INTO countries (country_id, country_name, region_id)
VALUES ('AT', 'Austria', 1);
```

Verändern von Daten

```
UPDATE countries
SET region_id = 2
WHERE country_name = 'Austria';
```

Löschen von Daten

```
DELETE
FROM countries
WHERE country_name = 'Austria';
```

MySQL Funktionen

- Spaltennamen der Zielrelation definieren mit AS

BSP:

```
SELECT salary AS Gehalt
FROM employees
```

MySQL Funktionen

- Durchschnitt mit `AVG()`

BSP:

```
SELECT AVG(salary) AS Durchschnittsgehalt
FROM employees
```

MySQL Funktionen

- Summenbildung mit `SUM()`

BSP:

```
SELECT SUM(salary), job_id
FROM employees
GROUP BY job_id
ORDER BY SUM(salary) DESC;
```

MySQL Funktionen

- Zeichenketten zusammenführen mit `CONCAT()` / `CONCAT_WS()`

BSP:

```
SELECT CONCAT(first_name, ' ', last_name) AS Name
FROM employees

SELECT CONCAT_WS(' ', first_name, last_name) AS Name
FROM employees
```

Grundlagen von JOINS

JOIN Verbindet zwei oder mehr Relationen miteinander.

Abfrage von Daten über zwei oder mehr Tabellen

Grundlagen von JOINS

CROSS JOIN - Jede Zeile von R1 verbunden mit jeder Zeile von R2

INNER JOIN - Verbindet alle Zeilen von R1 und R2 miteinander, wo ein Match gefunden wird.

OUTER JOIN - Verbindet alle Zeilen von R1 und R2 miteinander, wo ein Match gefunden wird. Wo keiner gefunden wird, wird der Rest mit NULL aufgefüllt.

LEFT JOIN - Jede Zeile von R1 verbunden mit dazupassenden Zeilen von R2

RIGHT JOIN - Jede Zeile von R2 verbunden mit dazupassenden Zeilen von R1

NATURAL JOIN - Natürlicher Verbund von R1 und R2 bei gleichnamiger Spalte

Grundlagen von JOINS

```
SELECT employees.last_name, departments.department_name
FROM employees
JOIN departments
ON employees.department_id = departments.department_id;
```

Grundlagen von JOINS

```
SELECT department_name, postal_code, city, country_name
FROM departments
JOIN locations ON departments.location_id = locations.location_id
JOIN countries ON locations.country_id = countries.country_id;
```

Grundlagen von JOINS

```
SELECT e2.last_name AS Manager, e1.last_name AS Unterstellter
FROM employees e1
JOIN employees e2
ON e1.manager_id = e2.employee_id
ORDER BY Manager;
```

Mengenoperationen

MINUS Ergebnisrelation A minus Ergebnisrelation B

UNION Ergebnisrelation A gemeinsam mit Ergebnisrelation B

INTERSECT Schnittmenge aus Ergebnisrelation A und Ergebnisrelation B

Mengenoperationen - UNION

```
SELECT job_id, department_id
FROM employees
WHERE department_id = 10
UNION
SELECT job_id, department_id
FROM  employees
WHERE department_id = 20
```

(NICHT MySQL) Mengenoperationen - Minus

```
SELECT department_id
FROM departments
MINUS
SELECT department_id
FROM  employees
WHERE job_id = 'ST_CLERK';
```

(NICHT MySQL) Mengenoperationen - INTERSECT

```
SELECT employee_id, job_id
FROM employees
INTERSECT
SELECT employee_id, job_id
FROM  job_history
```

Import
Export