Run ID,Agent Name,Task,Result,Timestamp

a6ee8830-c049-4a6f-9907-95ac7cfdfef6,Turnover-Analysis-Agent,Develop a sql schema to detect in aws data lake who is going to turn over and churn.,"**Employee Turnover Detection SQL Schema**

==========

Overview

The following SQL schema is designed to detect employee turnover and churn in an AWS data lake. It combines data from various sources to identify patterns and trends that may indicate a higher likelihood of turnover.

Table 1: Employee Information

```sql
CREATE TABLE employee_info (
  employee_id INT PRIMARY KEY,
  name VARCHAR(255),
  department VARCHAR(255),
  job_title VARCHAR(255),
  hire_date DATE,
  termination_date DATE
);
```

Table 2: Performance Metrics

```sql
CREATE TABLE performance_metrics (
  employee_id INT,
  performance_period DATE,
  performance_rating DECIMAL(3, 2),
  PRIMARY KEY (employee_id, performance_period),
  FOREIGN KEY (employee_id) REFERENCES employee_info(employee_id)
);
```

Table 3: Engagement Metrics

```sql
CREATE TABLE engagement_metrics (
  employee_id INT,
  engagement_period DATE,
  engagement_score DECIMAL(3, 2),
  PRIMARY KEY (employee_id, engagement_period),
  FOREIGN KEY (employee_id) REFERENCES employee_info(employee_id)
);
```

Table 4: Salary and Benefits

```sql
CREATE TABLE salary_and_benefits (
  employee_id INT,
  salary DECIMAL(10, 2),
  benefits DECIMAL(10, 2),
  PRIMARY KEY (employee_id),
  FOREIGN KEY (employee_id) REFERENCES employee_info(employee_id)
);
```

Table 5: Time Off and Leave

```sql
CREATE TABLE time_off_and_leave (
  employee_id INT,
  time_off_date DATE,
  leave_type VARCHAR(255),
  PRIMARY KEY (employee_id, time_off_date),
  FOREIGN KEY (employee_id) REFERENCES employee_info(employee_id)
);
```

Table 6: Turnover Indicators

```sql
CREATE TABLE turnover_indicators (
  employee_id INT,
  turnover_indicator VARCHAR(255),
  indicator_value DECIMAL(3, 2),
  PRIMARY KEY (employee_id, turnover_indicator),
  FOREIGN KEY (employee_id) REFERENCES employee_info(employee_id)
);
```

Table 7: Turnover Labels

```sql
CREATE TABLE turnover_labels (
  employee_id INT,
  turnover_label VARCHAR(255),
  PRIMARY KEY (employee_id),
  FOREIGN KEY (employee_id) REFERENCES employee_info(employee_id)
);
```

SQL Views for Turnover Detection

View 1: Employee Turnover Risk Score

```sql
CREATE VIEW employee_turnover_risk_score AS
SELECT
  ei.employee_id,
  ei.name,
  SUM(CASE WHEN tm.indicator_value > 0.5 THEN 1 ELSE 0 END) AS risk_score
FROM
  employee_info ei
  LEFT JOIN turnover_indicators ti ON ei.employee_id = ti.employee_id
  LEFT JOIN turnover_metrics tm ON ti.turnover_indicator = tm.turnover_indicator
GROUP BY
  ei.employee_id, ei.name;
```

View 2: Employee Turnover Probability

```sql
CREATE VIEW employee_turnover_probability AS
SELECT
  ei.employee_id,
  ei.name,
```

```
  (CASE
    WHEN tm.risk_score > 0.7 THEN 'High'
    WHEN tm.risk_score > 0.4 THEN 'Medium'
    ELSE 'Low'
  END) AS turnover_probability
FROM
  employee_info ei
  LEFT JOIN employee_turnover_risk_score tm ON ei.employee_id = tm.employee_id;
```

AWS Glue ETL Script

Create an AWS Glue ETL script to populate the turnover indicators table based on the employee information and performance metrics.

```python
import sys
from awsglue.transforms import *
from awsglue.utils import getResolvedOptions
from pyspark.context import SparkContext
from awsglue.context import GlueContext
from awsglue.job import Job

glueContext = GlueContext(SparkContext.getOrCreate())
spark = glueContext.spark_session
```

```
job = Job(glueContext)

# Read employee information and performance metrics
employee_info = glueContext.create_dynamic_frame.from_catalog(database='employee_database', table_name='employee_info')
performance_metrics = glueContext.create_dynamic_frame.from_catalog(database='employee_database', table_name='performance_metrics')

# Calculate turnover indicators
turnover_indicators = employee_info.join(performance_metrics, employee_info.employee_id == performance_metrics.employee_id)
turnover_indicators = turnover_indicators.withColumn('turnover_indicator', when(performance_metrics.performance_rating < 0.5, 'Low Performance').otherwise('High Performance'))

# Write turnover indicators to database
glueContext.write_dynamic_frame.from_catalog(frame=turnover_indicators, database='employee_database', table_name='turnover_indicators')

job.commit()
```

Turnover Detection Dashboard

Create a dashboard in Amazon QuickSight to visualize the employee turnover risk score and probability. Use the employee turnover risk score view and employee turnover probability view as data sources.

* **Risk Score Chart**: A bar chart displaying the employee turnover risk score for each employee.
* **Turnover Probability Chart**: A pie chart displaying the distribution of employee turnover probability.
* **Employee Turnover Table**: A table displaying the employee information, turnover risk score, and turnover probability.

By using this SQL schema, AWS Glue ETL script, and Amazon QuickSight dashboard, you can detect employee turnover and churn in your organization and take proactive measures to reduce turnover risk.",2024-10-17T19:24:23.054698
a6ee8830-c049-4a6f-9907-95ac7cfdfef6,Hiring-Trends-Analysis-Agent,Develop a sql schema to detect in aws data lake who is going to turn over and churn.,"**Predicting Employee Turnover and Churn using AWS Data Lake and SQL**

To detect employees who are likely to turn over and churn, we'll design a SQL schema to analyze relevant data in an AWS Data Lake. We'll use a combination of employee, performance, and engagement metrics to identify potential leavers.

Schema Design:

```sql
CREATE TABLE employees (
  employee_id VARCHAR(255) PRIMARY KEY,
```

```
  name VARCHAR(255),
  department VARCHAR(255),
  job_title VARCHAR(255),
  hire_date DATE,
  tenure INT,
  salary DECIMAL(10, 2),
  performance_rating DECIMAL(3, 2)
);

CREATE TABLE performance_data (
  employee_id VARCHAR(255),
  review_date DATE,
  performance_rating DECIMAL(3, 2),
  promotion_status VARCHAR(255),
  FOREIGN KEY (employee_id) REFERENCES employees(employee_id)
);

CREATE TABLE engagement_survey (
  employee_id VARCHAR(255),
  survey_date DATE,
  engagement_score DECIMAL(3, 2),
  satisfaction_score DECIMAL(3, 2),
  intention_to_leave_score DECIMAL(3, 2),
  FOREIGN KEY (employee_id) REFERENCES employees(employee_id)
);
```

```
CREATE TABLE turnover_data (
  employee_id VARCHAR(255),
  turnover_date DATE,
  reason_for_leaving VARCHAR(255),
  FOREIGN KEY (employee_id) REFERENCES employees(employee_id)
);

CREATE TABLE churn_prediction (
  employee_id VARCHAR(255),
  churn_probability DECIMAL(3, 2),
  churn_risk VARCHAR(255),
  FOREIGN KEY (employee_id) REFERENCES employees(employee_id)
);
```

Key Features:

1. **Employee Demographics**: The `employees` table stores basic employee information, such as name, department, job title, hire date, and salary.
2. **Performance Metrics**: The `performance_data` table captures employee performance ratings, review dates, and promotion status.
3. **Engagement Survey Data**: The `engagement_survey` table stores employee engagement scores, satisfaction scores, and intention to leave scores.
4. **Turnover Data**: The `turnover_data` table tracks turnover dates and reasons for leaving.
5. **Churn Prediction**: The `churn_prediction` table will store the predicted churn probability and risk for each employee.

Predicting Churn:

To predict churn, we'll use a combination of machine learning algorithms and data analysis techniques. We'll consider factors such as:

* **Tenure**: Employees with shorter tenure are more likely to leave.
* **Performance Rating**: Employees with lower performance ratings may be at a higher risk of leaving.
* **Engagement Score**: Employees with lower engagement scores may be more likely to leave.
* **Intention to Leave Score**: Employees with higher intention to leave scores are more likely to leave.

We'll use SQL queries to analyze the data and identify correlations between these factors and churn.

Example Query:

```sql
WITH churn_data AS (
  SELECT
    e.employee_id,
    e.tenure,
    p.performance_rating,
    es.engagement_score,
    es.intention_to_leave_score,
```

```
    CASE
      WHEN t.turnover_date IS NOT NULL THEN 1
      ELSE 0
    END AS churn
  FROM
    employees e
  LEFT JOIN
    performance_data p ON e.employee_id = p.employee_id
  LEFT JOIN
    engagement_survey es ON e.employee_id = es.employee_id
  LEFT JOIN
    turnover_data t ON e.employee_id = t.employee_id
)
SELECT
  churn_data.*,
    ROW_NUMBER() OVER (PARTITION BY churn_data.churn ORDER BY churn_data.tenure,
churn_data.performance_rating,                churn_data.engagement_score,
churn_data.intention_to_leave_score) AS churn_rank
FROM
  churn_data
ORDER BY
  churn_rank;
```

Next Steps:

1. **Data Analysis**: Analyze the data to identify correlations between employee demographics, performance metrics, engagement survey data, and churn.
2. **Machine Learning**: Train machine learning models to predict churn probability based on the analyzed data.
3. **Deployment**: Deploy the churn prediction model in the AWS Data Lake, and use the predicted churn probabilities to inform HR strategies and talent management decisions.

By using this schema and analyzing the data, you'll be able to identify employees who are likely to turn over and churn, enabling your organization to take proactive steps to retain top talent and reduce turnover costs.",2024-10-17T19:24:23.054698
a6ee8830-c049-4a6f-9907-95ac7cfdfef6,Top-Talent-Alert-Agent,Develop a sql schema to detect in aws data lake who is going to turn over and churn.,"**Turnover Risk Detection SQL Schema**
==========

Overview

This SQL schema is designed to detect turnover risk in an AWS data lake. It leverages various data points to identify high-performing individuals and teams who are at risk of leaving the company.

Database Schema

The following tables are used in the schema:

1. Employee Table

```sql
CREATE TABLE employees (
  employee_id INT PRIMARY KEY,
  name VARCHAR(255) NOT NULL,
  job_title VARCHAR(255) NOT NULL,
  department VARCHAR(255) NOT NULL,
  hire_date DATE NOT NULL,
  salary DECIMAL(10, 2) NOT NULL
);
```

2. Performance Table

```sql
CREATE TABLE performance (
  performance_id INT PRIMARY KEY,
  employee_id INT NOT NULL,
  rating DECIMAL(3, 2) NOT NULL,
  review_date DATE NOT NULL,
  FOREIGN KEY (employee_id) REFERENCES employees(employee_id)
);
```

3. Attendance Table

```sql
CREATE TABLE attendance (
  attendance_id INT PRIMARY KEY,
  employee_id INT NOT NULL,
  date DATE NOT NULL,
  hours_worked DECIMAL(5, 2) NOT NULL,
  FOREIGN KEY (employee_id) REFERENCES employees(employee_id)
);
```

4. Projects Table

```sql
CREATE TABLE projects (
  project_id INT PRIMARY KEY,
  project_name VARCHAR(255) NOT NULL,
  start_date DATE NOT NULL,
  end_date DATE NOT NULL
);
```

5. Project Assignments Table

```sql
CREATE TABLE project_assignments (
  project_assignment_id INT PRIMARY KEY,
```

```
  employee_id INT NOT NULL,
  project_id INT NOT NULL,
  join_date DATE NOT NULL,
  leave_date DATE NOT NULL,
  FOREIGN KEY (employee_id) REFERENCES employees(employee_id),
  FOREIGN KEY (project_id) REFERENCES projects(project_id)
);
```

6. Turnover Risk Table

```sql
CREATE TABLE turnover_risk (
  turnover_risk_id INT PRIMARY KEY,
  employee_id INT NOT NULL,
  risk_score DECIMAL(3, 2) NOT NULL,
  risk_level VARCHAR(255) NOT NULL,
  FOREIGN KEY (employee_id) REFERENCES employees(employee_id)
);
```

Derived Tables

The following derived tables are used to calculate turnover risk:

1. Average Performance Table

```sql
CREATE TABLE avg_performance (
  employee_id INT PRIMARY KEY,
  avg_rating DECIMAL(3, 2) NOT NULL,
  FOREIGN KEY (employee_id) REFERENCES employees(employee_id)
);

INSERT INTO avg_performance (employee_id, avg_rating)
SELECT employee_id, AVG(rating) AS avg_rating
FROM performance
GROUP BY employee_id;
```

2. Average Attendance Table

```sql
CREATE TABLE avg_attendance (
  employee_id INT PRIMARY KEY,
  avg_hours_worked DECIMAL(5, 2) NOT NULL,
  FOREIGN KEY (employee_id) REFERENCES employees(employee_id)
);

INSERT INTO avg_attendance (employee_id, avg_hours_worked)
SELECT employee_id, AVG(hours_worked) AS avg_hours_worked
```

FROM attendance

GROUP BY employee_id;

```

### **3. Project Participation Table**

```sql
CREATE TABLE project_participation (
  employee_id INT PRIMARY KEY,
  num_projects INT NOT NULL,
  FOREIGN KEY (employee_id) REFERENCES employees(employee_id)
);

INSERT INTO project_participation (employee_id, num_projects)
SELECT employee_id, COUNT(project_id) AS num_projects
FROM project_assignments
GROUP BY employee_id;
```

Turnover Risk Scoring

The turnover risk score is calculated based on the following factors:

* Average performance rating
* Average attendance hours worked
```

* Number of projects participated in

The scoring system uses a weighted average of these factors to determine the overall turnover risk score.

```sql
INSERT INTO turnover_risk (turnover_risk_id, employee_id, risk_score, risk_level)
SELECT
  employee_id,
  (0.4 * avg_rating + 0.3 * avg_hours_worked + 0.3 * num_projects) AS risk_score,
  CASE
    WHEN (0.4 * avg_rating + 0.3 * avg_hours_worked + 0.3 * num_projects) < 0.5 THEN 'Low'
    WHEN (0.4 * avg_rating + 0.3 * avg_hours_worked + 0.3 * num_projects) < 0.8 THEN 'Medium'
    ELSE 'High'
  END AS risk_level
FROM
  avg_performance
  INNER JOIN avg_attendance ON avg_performance.employee_id = avg_attendance.employee_id
        INNER JOIN project_participation ON avg_performance.employee_id = project_participation.employee_id;
```

Alert System

The alert system is triggered when an employee's turnover risk score exceeds a certain threshold.

```sql
CREATE PROCEDURE trigger_alert()
BEGIN
  DECLARE done INT DEFAULT FALSE;
  DECLARE employee_id INT;
  DECLARE risk_score DECIMAL(3, 2);

  DECLARE cur1 CURSOR FOR
    SELECT employee_id, risk_score
    FROM turnover_risk
    WHERE risk_score > 0.8;

  DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

  OPEN cur1;

  read_loop: LOOP
    FETCH cur1 INTO employee_id, risk_score;
    IF done THEN
      LEAVE read_loop;
    END IF;

    -- Trigger alert for employee with high turnover risk score
    INSERT INTO alerts (employee_id, alert_message)
    VALUES (employee_id, 'High turnover risk detected');
```

```
  END LOOP;

  CLOSE cur1;
END;
```

Reports and Recommendations

Regular reports are generated to track turnover risk trends and provide recommendations for retention strategies.

```sql
CREATE PROCEDURE generate_report()
BEGIN
  SELECT
    department,
    COUNT(employee_id) AS num_employees,
    AVG(risk_score) AS avg_risk_score
  FROM
    employees
    INNER JOIN turnover_risk ON employees.employee_id = turnover_risk.employee_id
  GROUP BY
    department;

  -- Provide recommendations for retention strategies based on turnover risk scores
```

```
  SELECT
    employee_id,
    name,
    job_title,
    department,
    risk_score,
    CASE
      WHEN risk_score < 0.5 THEN 'No action required'
      WHEN risk_score < 0.8 THEN 'Monitor employee performance and provide feedback'
      ELSE 'Develop retention plan and provide incentives'
    END AS recommendation
  FROM
    employees
    INNER JOIN turnover_risk ON employees.employee_id = turnover_risk.employee_id;
END;
```",2024-10-17T19:24:23.054698
a6ee8830-c049-4a6f-9907-95ac7cfdfef6,Data-Integration-Agent,Develop a sql schema to detect in aws data lake who is going to turn over and churn.,"**SQL Schema for Employee Turnover and Churn Analysis in AWS Data Lake**

To detect employee turnover and churn, we will design a SQL schema that captures key employee, hiring, and workforce data. We will use Amazon Redshift as our data warehousing solution in AWS Data Lake.

Schema Name: `hr_analytics`

Tables:

1. `employees`

Column Name	Data Type	Description
`employee_id`	`int`	Unique identifier for each employee
`name`	`varchar(255)`	Employee name
`email`	`varchar(255)`	Employee email
`department`	`varchar(100)`	Employee department
`job_title`	`varchar(100)`	Employee job title
`hire_date`	`date`	Employee hire date
`termination_date`	`date`	Employee termination date
`termination_reason`	`varchar(255)`	Reason for termination (optional)

2. `hiring_data`

Column Name	Data Type	Description
`hiring_id`	`int`	Unique identifier for each hiring event
`employee_id`	`int`	Foreign key referencing `employees.employee_id`
`job_title`	`varchar(100)`	Job title for the hiring event
`hiring_date`	`date`	Date of the hiring event
`source`	`varchar(100)`	Source of the hiring event (e.g., referral, job posting)

3. `workforce_data`

| Column Name | Data Type | Description |

| --- | --- | --- |

| `workforce_id` | `int` | Unique identifier for each workforce record |

| `employee_id` | `int` | Foreign key referencing `employees.employee_id` |

| `date` | `date` | Date of the workforce record |

| `absent` | `boolean` | Whether the employee was absent on that date |

| `late` | `boolean` | Whether the employee was late on that date |

| `productivity` | `float` | Employee productivity score (optional) |

4. `turnover_data`

| Column Name | Data Type | Description |

| --- | --- | --- |

| `turnover_id` | `int` | Unique identifier for each turnover event |

| `employee_id` | `int` | Foreign key referencing `employees.employee_id` |

| `turnover_date` | `date` | Date of the turnover event |

| `turnover_reason` | `varchar(255)` | Reason for turnover (optional) |

5. `churn_data`

| Column Name | Data Type | Description |

| --- | --- | --- |

| `churn_id` | `int` | Unique identifier for each churn event |

| `employee_id` | `int` | Foreign key referencing `employees.employee_id` |

| `churn_date` | `date` | Date of the churn event |

| `churn_reason` | `varchar(255)` | Reason for churn (optional) |

Relationships:

* An employee can have multiple hiring events (one-to-many).
* An employee can have multiple workforce records (one-to-many).
* An employee can have one turnover event (one-to-one).
* An employee can have one churn event (one-to-one).

ETL Process:

1. Extract data from various sources (e.g., HR systems, time-tracking software).
2. Transform data into a standardized format.
3. Load data into the respective tables in the `hr_analytics` schema.

Optimizations:

1. Create indexes on frequently queried columns (e.g., `employee_id`, `date`).
2. Use partitioning on large tables (e.g., `workforce_data`) to improve query performance.
3. Use Amazon Redshift's built-in data compression to reduce storage costs.

Data Security and Privacy:

1. Implement row-level security to restrict access to sensitive data (e.g., termination reasons).
2. Use encryption to protect data at rest and in transit.
3. Ensure compliance with relevant data privacy regulations (e.g., GDPR, CCPA).

Data Dictionary and Documentation:

1. Create a data dictionary to document each table, column, and data type.
2. Provide detailed documentation on the ETL process, data transformations, and data quality checks.
3. Maintain version control and tracking of schema changes.

By following this schema and implementing the necessary ETL processes, data security measures, and documentation, we can create a robust and scalable data analytics platform to detect employee turnover and churn in AWS Data Lake.",2024-10-17T19:24:23.054698 a6ee8830-c049-4a6f-9907-95ac7cfdfef6,Predictive-Analytics-Management-Agent,Develop a sql schema to detect in aws data lake who is going to turn over and churn.,"**Predictive Analytics Schema for Turnover and Churn Detection**

Overview

To detect potential turnover and churn in our organization, we will create a SQL schema in our AWS Data Lake using Amazon Redshift. This schema will integrate data from various sources, including HR systems, employee engagement surveys, and performance metrics.

Schema Design

Table 1: Employee Master Data

```sql
CREATE TABLE employee_master (
    employee_id INT PRIMARY KEY,
    employee_name VARCHAR(255),
    job_title VARCHAR(255),
    department VARCHAR(255),
    hire_date DATE,
    termination_date DATE,
    tenure INT
);
```

Table 2: Employee Engagement Surveys

```sql
CREATE TABLE employee_engagement_surveys (
    survey_id INT PRIMARY KEY,
    employee_id INT,
    survey_date DATE,
    engagement_score DECIMAL(3, 2),
    satisfaction_score DECIMAL(3, 2),
    likelihood_to_recommend_score DECIMAL(3, 2),
    FOREIGN KEY (employee_id) REFERENCES employee_master(employee_id)
);
```

Table 3: Performance Metrics

```sql
CREATE TABLE performance_metrics (
    performance_id INT PRIMARY KEY,
    employee_id INT,
    review_date DATE,
    performance_rating VARCHAR(255),
    goals_met DECIMAL(3, 2),
    FOREIGN KEY (employee_id) REFERENCES employee_master(employee_id)
);
```

Table 4: Turnover and Churn Predictions

```sql
CREATE TABLE turnover_churn_predictions (
    prediction_id INT PRIMARY KEY,
    employee_id INT,
    prediction_date DATE,
    turnover_probability DECIMAL(5, 4),
    churn_probability DECIMAL(5, 4),
    predicted_turnover_date DATE,
    predicted_churn_date DATE,
    FOREIGN KEY (employee_id) REFERENCES employee_master(employee_id)
```

);

```

### **Table 5: Alert System**

```sql
CREATE TABLE alert_system (
    alert_id INT PRIMARY KEY,
    employee_id INT,
    alert_date DATE,
    alert_type VARCHAR(255),
    alert_message VARCHAR(255),
    FOREIGN KEY (employee_id) REFERENCES employee_master(employee_id)
);
```

Predictive Modeling

To detect potential turnover and churn, we will use a machine learning model that combines data from the `employee_engagement_surveys`, `performance_metrics`, and `employee_master` tables. We will use a supervised learning approach, where we train the model on historical data to predict future outcomes.

Example Query

```

To identify employees who are likely to turn over or churn, we can use the following query:

```sql
SELECT
    e.employee_id,
    e.employee_name,
    t.turnover_probability,
    t.churn_probability
FROM
    employee_master e
JOIN
    turnover_churn_predictions t ON e.employee_id = t.employee_id
WHERE
    t.turnover_probability > 0.5 OR t.churn_probability > 0.5;
```

This query joins the `employee_master` and `turnover_churn_predictions` tables on the `employee_id` column and selects employees with a turnover or churn probability greater than 0.5.

Alert System

To notify stakeholders of potential turnover and churn, we can create an alert system that triggers when an employee's turnover or churn probability exceeds a certain threshold.

```sql
INSERT INTO alert_system (alert_id, employee_id, alert_date, alert_type, alert_message)
SELECT
```

```
    NEXTVAL(alert_id_seq),
    t.employee_id,
    CURRENT_DATE,
    'Turnover' AS alert_type,
    'Employee is at risk of turnover' AS alert_message
FROM
    turnover_churn_predictions t
WHERE
    t.turnover_probability > 0.7;
```

This query inserts a new alert into the `alert_system` table when an employee's turnover probability exceeds 0.7.

Compliance and Ethics

To ensure compliance with relevant regulations and ethical use of data, we will:

* Use pseudonymized data to protect employee identities
* Implement data access controls to restrict access to authorized personnel
* Use transparent and explainable machine learning models to avoid bias and ensure fairness
* Regularly review and update our predictive models to ensure accuracy and relevance",2024-10-17T19:24:23.054698